

ALAGAPPA UNIVERSITY



[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle and Graded as Category–I University by MHRD-UGC]

(A State University Established by the Government of Tamil Nadu)

KARAIKUDI – 630 003

Directorate of Distance Education

B.C.A.

IV - Semester

101 43

INTERNET

AND

JAVA PROGRAMMING

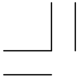
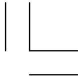
Authors:
Rajneesh Agrawal, Senior Scientist, Department of Information Technology, Government of India
Units: (1.2, 1.2.2, 1.2.4, 1.3-1.4, 2.2-2.4)
Rohit Khurana, CEO, ITL Education Solutions Ltd., 2nd Floor, GD-ITL Tower, Netaji Subhash Place, Pitampura, New Delhi
Units: (4.4, 4.5-4.6, 5, 6.0-6.3, 6.5-6.11, 7.0-7.6, 7.9-7.10, 8.2-8.5, 9, 10, 11.3-11.4, 13, 14)
Kalapatapu Kalyani, Asstt. Professor, Matrusri Institute of Post Graduate Studies, Affiliated to Osmania University & Recognized by AICTE, Hyderabad
Units: (4.2, 6.4, 7.8)
Vikas® Publishing House: Units: (1.0-1.1, 1.2.1, 1.2.3, 1.5-1.9, 2.0-2.1, 2.5-2.10, 3, 4.0-4.1, 4.3, 4.7-4.11, 7.7, 7.11-7.15, 8.0-8.1, 8.6-8.12, 11.0-11.2, 11.5-11.9, 12)



Vikas® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

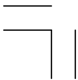
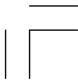
Work Order No.AU/DDE/DE12-15/Printing of Course Material/2020 dated 28.02.2020, Copies - 500

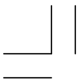
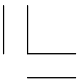



SYLLABI-BOOK MAPPING TABLE

Internet and Java Programming

Syllabi	Mapping in Book
BLOCK I: INTRODUCTION **Unit 1:** Basic Internet Concepts: Connecting to the Internet - Domain Name System - E-Mail **Unit 2:** The World Wide Web - Internet Search Engines - Web Browsers - Chatting and Conferencing on the Internet **Unit 3:** Online Chatting -Messaging - Usenet Newsgroup - Internet Relay Chat (IRC) - FTP - Telnet.	**Unit 1:** Basic Internet Concepts **(Pages 1-38);** **Unit 2:** The World Wide Web **(Pages 39-54);** **Unit 3:** Internet Chats, Internet Relay Chat (IRC), FTP and Telnet **(Pages 55-66)**
BLOCK 2 : FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING **Unit 4:** Basic Concepts of OOP - Benefits - Applications, Java Evolution: Features - How Java Differs from C and C++, Java and Internet- Java Support System - Java Environment **Unit 5:** Overview of Java Language - Introduction - Simple Java Program - Comments - Java Program Structure - Tokens - Java Statements - Implementing a Java Program - JVM - Command Line Arguments. Constants - Variables - Data Types - Type Casting. **Unit 6:** Operators and Expressions: Arithmetic Operators - Relational, Logical, Assignment, Increment and Decrement, Conditional, Bitwise, Special Operators - Arithmetic Expressions, Evaluation of Expression - Precedence of Arithmetic Operators - Type Conversions - Operator Precedence and Associativity - Mathematical Functions. Decision Making and Branching: If -If…..Else - Nesting of If……. Else - Else If- Switch. Decision Making and Looping: While - Do - For - Jump in Loops - Labeled Loops.	**Unit 4:** Basic Concepts of OOP **(Pages 67-86);** **Unit 5:** Overview of Java Language **(Pages 87-110);** **Unit 6:** Operators and Expressions, Decision-Making and Branching **(Pages 111-140)**
BLOCK 3 : CLASSES, OBJECTS AND METHODS **Unit 7:** Class: Defining a Class -Fields -Methods -Creating Objects - Accessing Class Members - Constructors - Methods Overloading - Static Members -Nesting of Methods - Inheritance - Overriding Methods -Final Variables-Classes -Methods **Unit 8:** Arrays, Strings and Vectors : One Dimensional Arrays - Creating of Array - Two Dimensional Arrays- Strings -Vectors - Wrapper Classes - Enumerated Types - Interfaces: Multiple Inheritance **Unit 9:** Packages: Defining Interface -Extending Interfaces - Implementing Interfaces - Putting Classes Together	**Unit 7:** Classes in Java **(Pages 141-170);** **Unit 8:** Arrays, Strings and Vectors **(Pages 171-192);** **Unit 9:** Packages **(Pages 193-222)**

BLOCK 4: MULTITHEREADING, EXCEPTION AND APPLETS

Unit 10: Multithreaded Programming - Creating Threads - Extending the Thread Class - Stopping and Blocking a Thread - Life Cycle of a Thread - Using Thread Methods - Thread Exceptions - Priority - Synchronization - Implementing the 'Runnable' Interface

Unit 11: Managing Error and Exceptions: Types of Errors -Exceptions -Syntax of Exception Handling Code - Multiple Catch Statements - Using Finally Statement - Throwing Our Own Exceptions - Using Exceptions for Debugging - Graphics Programming: The Graphics Class - Lines and Rectangles - Circles and Ellipses - Drawing Arcs - Drawing Polygons - Line Graphs - Using Control Loops in Applets - Drawing Bar Charts

Unit 12: Applet Programming: How Applets Differ from Applications - Preparing to Write Applets - Building Applet Code - Applet Life Cycle - Creating an Executable Applet - Designing a Web Page - Applet Tag - Adding Applet to HTML File - Running the Applet - Passing Parameters to Applets - Displaying Numerical Values - Getting Input From The User

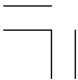
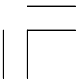
---

BLOCK 5 : MANAGING INPUT/OUTPUT FILES IN JAVA

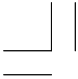
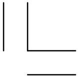
Unit 13: Introduction - Concept of Streams -Stream Classes - Byte Stream Classes - Character Stream

Unit 14: I/O Classes: - Using Stream -Using The File Class -Input / Output Exceptions - Creation of Files - Reading / Writing Characters - Reading Writing Bytes Random Access Files - Interactive Input and Output -Other Stream Classes

---

CONTENTS

INTRODUCTION

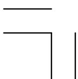
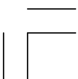
NOTES

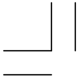
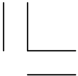
The Internet is the global system of interconnected computer networks that uses the Internet protocol suite (TCP/IP) to communicate between networks and devices. It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries a vast range of information resources and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), electronic mail, telephony, and file sharing. Java is a multi-threaded object-oriented language that incorporates many Internet features, such as security and the ability to stream data from a uniform resources.

Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers Write Once, Run Anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java Virtual Machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but it has fewer low-level facilities than either of them. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle) and released in 1995 as a core component of Sun Microsystems' Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses.

Fundamentally, Java inherited most of the concepts and fundamentals from languages like C and C++. The inheritance of OOPs and programming concepts aided in the early conception of the Java language by C and C++ users. In addition, Java allows developers to create two types of programs, i.e., applications and applets. Applications are programs that run on user operating systems and perform tasks based on the input provided by the user and the processing logic defined by the programmer. Applets are small applications that can be transmitted over the Internet and are executed on the user computer by using a Java compatible Web browser.

This book, *Internet and Java Programming*, is divided into five blocks that are further divided into fourteen units which will help you understand the basic Internet concepts, connecting to the Internet, Domain Name System (DNS), E-mail, the World Wide Web (WWW), Internet search engines – web browsers – chatting and conferencing on the Internet, online chatting, messaging, Usenet Newsgroup, Internet Relay Chat (IRC), FTP, Telnet, basic concepts of OOP, Java evolution: features, how Java differs from C and C++, Java and the Internet, Java environment, Java language, simple Java Program, comments, tokens, Java

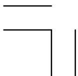
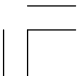



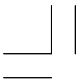



NOTES

statements, JVM, constants, variables, data types, type casting, operators and expressions, arithmetic operators, arithmetic expressions, evaluation of expression, precedence of arithmetic operators, operator precedence and associativity, decision-making and branching, looping, class, fields, methods, accessing class members, constructors, static members, inheritance, arrays, strings, vectors, wrapper classes, interfaces: multiple inheritance, packages, interface, multithreaded programming, threads, life cycle of a thread, managing error and exceptions, graphics programming, applet programming, applet life cycle, applet tag, passing parameters to applets, stream classes and I/O classes.

The book follows the Self-Instruction Mode or the SIM format wherein each unit begins with an 'Introduction' to the topic followed by an outline of the 'Objectives'. The content is presented in a simple, organized and comprehensive form interspersed with 'Check Your Progress' questions and answers for better understanding of the topics covered. A list of 'Key Words' along with a 'Summary' and a set of 'Self Assessment Questions and Exercises' is provided at the end of the each unit for effective recapitulation. Logically arranged topics, relevant solved examples and illustrations have been included for better understanding of the topics.




BLOCK - I
INTRODUCTION

NOTES

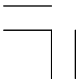
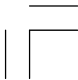
UNIT 1 BASIC INTERNET CONCEPTS

Structure

1.0 INTRODUCTION

The Internet is the global system of interconnected computer networks that uses the Internet Protocol Suite (TCP/IP) to communicate between networks and devices. It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries a vast range of information resources and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), electronic mail, telephony, and file sharing.

The origins of the Internet date back to the development of packet switching and research commissioned by the United States Department of Defense in the 1960s to enable time-sharing of computers. The primary precursor network, the ARPANET, initially served as a backbone for interconnection of regional academic and military networks in the 1970s. The linking of commercial networks and enterprises by the early 1990s marked the beginning of the transition to the modern Internet, and generated a sustained exponential growth as generations of institutional, personal, and mobile computers were connected to the network.

NOTES

The Internet has no single centralized governance in either technological implementation or policies for access and usage; each constituent network sets its own policies. The overreaching definitions of the two principal name spaces in the Internet, the Internet Protocol address (IP address) space and the Domain Name System (DNS), are directed by a maintainer organization, the Internet Corporation for Assigned Names and Numbers (ICANN).

Electronic mail (email or e-mail) is a method of exchanging messages in the form of 'Mail' between people using electronic devices. Ray Tomlinson is credited as the inventor of e-mail; in 1971, he developed the first system able to send mail between users on different hosts across the ARPANET, using the @ sign to link the user name with a destination server. By the mid-1970s, this was the form recognized as e-mail.

In this unit, you will study about the basic Internet concepts, connecting to the Internet, Domain Name System (DNS) and e-mail.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the significance of Internet concepts
- Define the importance of Internet
- Discuss how to get connected to the Internet
- Understand what Domain Name System (DNS) is
- Elaborate on e-mail concept

1.2 BASIC INTERNET CONCEPTS

The Internet, World Wide Web and Information Super Highway are terms which have the lives of millions of people all over the world. The widespread impact of Internet across the globe could not be possible without the development of Transmission Control Protocol/Internet Protocol (TCP/IP). This protocol suite is developed specifically for the Internet. The Information Technology revolution could not have been achieved without this boundless chain of networks. It has become a fundamental part of the lives of millions of people all over the world. All the aforesaid services, provide us the necessary backbone for information sharing in organizations and within common interest groups. That information may be in several forms. It can be notes and documents, data to be processed by another computer, files sent to colleagues, and even more exotic forms of data.

During late 60s and 70s, organizations were inundated with many different LAN and WAN technologies such as packet switching technology, collision-detection local area networks, hierarchical enterprise networks, and many other excellent technologies. The major drawbacks of all these technologies were that

they could not communicate with each other without expensive deployment of communications devices. These were not only expensive but also put users at the mercy of the monopoly of the vendor they were dealing with. Consequently, multiple networking models were available as a result of the research and development efforts made by many interest groups. This paved the way for development of another aspect of networking known as protocol layering. This allows applications to communicate with each other. A complete range of architectural models were proposed and implemented by various research teams and computer manufacturers. The result of this know-how is that today any group of users can find a physical network and an architectural model suitable for their specific needs. This includes cheap asynchronous lines with no other error recovery than a bit-per-bit parity function, through full-function wide area networks (public or private) with reliable protocols such as public packet switching networks or private SNA networks, to high-speed but limited-distance local area networks.

It is now evident that organizations or users are using different network technologies to connect computers over the network. The desire of sharing more and more information among homogeneous or heterogeneous interest groups motivated the researcher to device a technology whereby one group of users could extend its information system to another group who had a different network technology and different network protocols. This necessity was recognized in early 70s by a group of researchers in the United States of America (USA) who hit upon a new principle popularly known as Internetworking. Other organizations also became involved in this area of interconnecting networks, such as ITU-T (formerly CCITT) and ISO. All were trying to define a set of protocols, layered in a well-defined suite, so that applications would be able to communicate with each other, regardless of the underlying network technology and the operating systems where those applications run.

Internetworks

The availability of different operating systems, hardware platforms and the geographical dispersion of computing resources necessitated the need of networking in such a manner that computers of all sizes could communicate with each other, regardless of the vendor, the operating system, the hardware platform, or geographical proximity. Therefore, we may say that *Internetworking* is a scheme for interconnecting multiple networks of dissimilar technologies. To interconnect multiple networks of dissimilar technologies use both additional hardware and software. This additional hardware is positioned between networks and software on each attached computer. This system of interconnected networks is called an *Internetwork* or an *Internet*.

To develop standards for Internetworking, Defense Advanced Research Projects Agency (DARPA) funded research projects. ARPAnet, a project of DARPA, introduced the world of networking with protocol suite concepts such as layering, well before ISO's initiative in this direction. DARPA continued its

NOTES

research for an Internetworking protocol suite. This may be seen in the early NCP (Network Control Program) host-to-host protocol to the TCP/IP protocol suite, which took its current form around 1978. DARPA was well known for its pioneering of packet switching over radio networks and satellite channels and ARPAnet was declared an operational network with responsibility of administering it to Defense Communications Agency (DCA) in 1975. TCP/IP had not yet been developed.

ARPAnet was basically a network based on leased lines connected by special switching nodes, known as Internet Message Processors (IMP). Many researchers were involved in TCP/IP research by 1979. This motivated DARPA to form an informal committee to coordinate and guide the design of the communication protocols and architecture. The committee was called the Internet Control and Configuration Board (ICCB).

The first real implementation of the Internet was when DARPA converted the machines of its research network ARPAnet to use the new TCP/IP protocols. After this transition which started in 1980 and finished in 1983, DARPA demanded that all computers willing to connect to its ARPAnet must use TCP/IP. The US military adopted TCP/IP as standard protocol in 1983 and recommended that all networks connected to the ARPAnet conform to the new standards.

The success of ARPAnet was more than the expectations of its own founders and TCP/IP Internetworking became widespread. As a result, new wide area networks (WAN) were created in the USA and connected to ARPAnet using TCP/IP protocol. In turn, other networks in the rest of the world, not necessarily based on the TCP/IP protocols, were added to the set of interconnected networks. Computing facilities all over North America, Europe, Japan, and other parts of the world are currently connected to the Internet via their own sub-networks, constituting the world's largest network. In 1990, ARPAnet was eliminated, and the Internet was declared as the formal global network.

DARPA also funded a project to develop TCP/IP protocols for Berkeley UNIX on the VAX and to distribute the developed codes free of charge with their UNIX operating system. The first release of the Berkeley Software Distribution (BSD) to include the TCP/IP protocol set was made available in 1983 (4.2BSD). This led to the spread of TCP/IP among universities and research centers and has become the standard communications subsystem for all UNIX connectivity. There are many updated versions of BSD code available. These are 4.3BSD (1986), 4.3BSD Tahoe (1988), 4.3BSD Reno (1990) and 4.4BSD (1993).

Some examples of the different networks that have played key roles in this development are described below:

The Internet

The word Internet is a short form of a complete word Internetwork or interconnected network. Therefore, it can be said that the Internet is not a single

network, but a collection of networks. The commonality between them in order to communicate with each other is TCP/IP. The Internet consists of the following groups of networks:

NOTES

- **Backbones:** These are large networks that exist primarily to interconnect other networks. Some examples of backbones are NSFNET in the USA, EBONE in Europe and large commercial backbones.
- **Regional Networks:** These connect, for example, universities and colleges. ERNET (Education and Research Network) is an example in the Indian context.
- **Commercial Networks:** They provide access to the backbones to subscribers, and networks owned by commercial organizations for internal use and also have connections to the Internet. Mainly, Internet Service Providers come into this category.
- **Local Networks:** These are campus-wide university networks.

The networks connect users to the Internet using special devices that are called gateways or routers. These devices provide connection and protocol conversion of dissimilar networks to the Internet. Gateways or routers are responsible for routing data around the global network until they reach their ultimate destination as shown in Figure 1.1. The delivery of data to its final destination takes place based on some routing table maintained by router or gateways. These are mentioned at various places in this book as these are the fundamental devices to connect similar or dissimilar networks together.

Over time, TCP/IP defined several protocol sets for the exchange of routing information. Each set pertains to a different historic phase in the evolution of architecture of the Internet backbone.

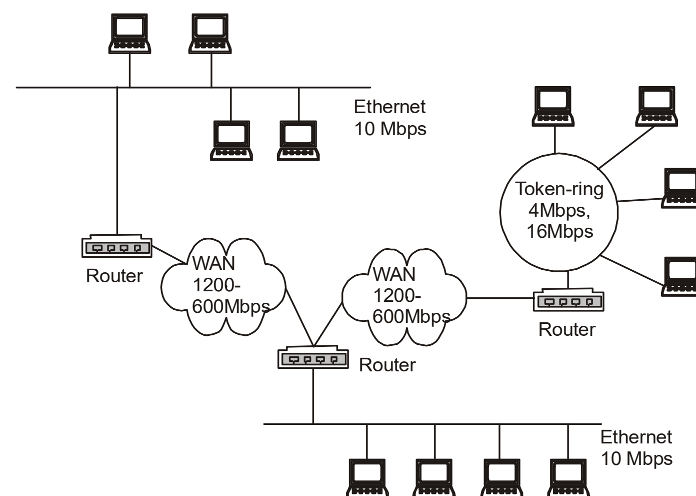


Fig. 1.1 *Local Area Networks Connected to the Internet via Gateways or Routers*

ARPAnet

ARPAnet was built by DARPA as described earlier. This initiated the packet switching technology in the world of networking and therefore is sometimes referred to as the "grand-daddy of packet networks". The ARPAnet was established in the late 60s for Department of Defense to accommodate research equipment on packet switching technology besides allowing resource sharing for the Department's contractors. This network includes research centres, some military bases and government locations. It soon became popular with researchers for collaboration through electronic mail and other services. ARPAnet marks the beginning of Internet.

ARPAnet provided interconnection of various packet-switching nodes (PSN) located across continental USA and Western Europe using 56 Kbps leased lines. ARPAnet provided connection to minicomputers running a protocol known as 1822 (after the number of a report describing it) and dedicated it to the packet-switching task. Each PSN had at least two connections to other PSNs (to allow alternate routing in case of circuit failure) and up to 22 ports for user computer connections. Later on, DARPA replaced the 1822 packet switching technology with the CCITT X.25 standard. The increase in data traffic made 56 Kbps capacity of the lines insufficient. ARPAnet has now been replaced with new technologies as backbone for the research side of the connected Internet.

1.2.1 Internet Services

The Internet is known as 'the Network of Networks'. It is like a phone system that connects almost anywhere around the world. It exchanges information and acts as global link between small regional networks. Internet services offer a gateway to a myriad of online databases, library catalogues and collections, and software and document archives, in addition to frequently used store-and-forward services, such as UserNet News and e-mail. The widely used Internet services are as follows:

E-Mail

E-mail is the prime Internet service that facilitates services to people or users across the world. Full Internet connectivity is not required for this. For example, an electronic address provides these services to FTP sites through which mail can be exchanged. Other Internet services, such as IP address resolver, Archie lookup, WHOIS service is done via e-mail.

The header and body of the message make an e-mail message. The header contains the information where the message is to be sent and the complete path for reaching the destination, date and return path. The body of the message is the actual message that has to be sent. The syntax of an e-mail address is user@ subdomain.subdomain.domain, e.g., **abc@gmail.com**. A service provider must be connected with leased line, dial-up or connection with any network for sending e-mail.

NOTES

File Transfer Protocol (FTP)

FTP is also prime Internet service that acts as protocol and transfers files over TCP/IP network (Internet, UNIT, etc.). Once HTML page is developed on a local machine for a website, it is first uploaded to the Web server through FTP. Local machine is the machine on which you are initially logged into. It includes functions to log on to the network, gives a list of directories and copies files. FTP transfer is possible by entering URL preceded with ftp:// within address bar of a web browser. The FTP operations can be performed by issuing FTP commands at the command prompt or by using FTP utility running under a graphical user interface on Windows OS. FTP tasks can be performed through a browser. For example, type an IE address bar URL as ftp:// to get ftp services. For example, ftp://YourLoginName@IPaddress.

The required steps used in connecting with FTP operations are as follows:

The local machine is connected with remote machine by typing 'ftp machinename'. The machinename is the full name, written as aaa.cs.state.edu, of the remote machine to which the local machine is to be connected. Basically, the machine name is the remote machine's full name. If the machine name is not available other option is taken as to type the 'ftp machinenumber' that demands the Net address of the remote machine, e.g., 129.15.0.11. The FTP responds to the users to enter their loginname and password. The anonymous ftp is used widely these days. Many computer systems provide this facility so that you can access the information of specific machine without creating an account on that particular machine. These types of services are provided by anonymous FTP. You need not to be registered user of the system. The anonymous FTP server contains relevant software, documents and files used to configure networks, graphics, images, songs, lyrics and other useful information. An electronic e-mail can be archived through the anonymous FTP. The ready information is stored in machines for any user across the Net who wants to get the required information.

Telnet

Telnet is used to connect remote network computers. It is the Internet service that executes commands on remote host as if you are going to log in locally. For this, the machine name and valid user name are required to be connected. The commands that are issued on telnet are as follows:

***Telnet Hostname*:** A connection to the host name is opened by this command. For example, issuing the command as 'telnet abc.maths.edu' with that machine which keeps the required information of abc.maths.edu site can connect you.

***Telnet Address*:** It gives the IP address of the connected host.

NOTES

Archie

If some programs are installed in a system unit and you want to know the availability of the program on the Internet, you can get to know the machine along with such programs via Archie. Basically, Archie is a program that searches files anywhere on the Net by filename. This facility is maintained by a database with the Internet sites accessible via anonymous ftp. The following Table 1.1 shows the various types of Archie servers:

Table 1.1 *Various Types of Archie Servers*

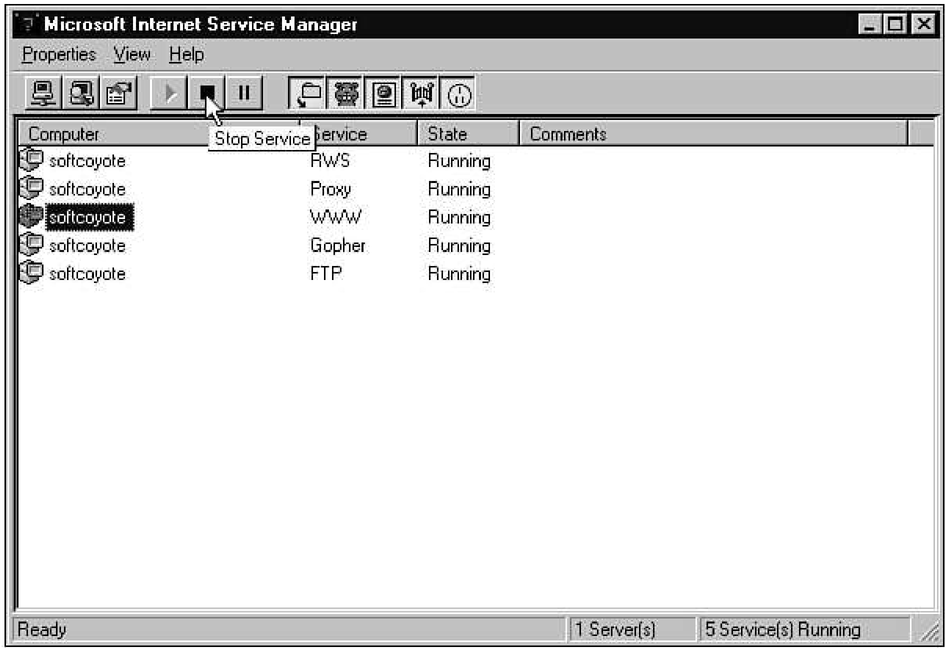
Name	IP Address
archie.rutgers.edu	128.6.18.15
archie.cs.mcgill.ca	132.206.51.250
archie.funet.fi	128.214.6.102
archie.rediris.es	130.206.1.2
archie.sura.net	192.239.16.13
archie.doc.ic.ac.uk	146.169.16.11

The Archie server can be accessed via Telnet, for example, 'Telnet archie.rutgers.edu'. For getting Archie server login to 'Archie'. It requires no password. You can look for files by its full name. For this, either 'set type exact' syntax is used or you can use 'set type sub' syntax. The 'set type sub' syntax is used if the required name of the file is known. The 'find file-name' syntax is also used to find the required file name.

Gopher

The Gopher protocol supports client–server software that searches files on the Internet. A Gopher client is required for validating and testing of Gopher publishing service. For example, WS Gopher 1.2 is available on the Internet as shareware. The server based text files are hierarchically organized and viewed by end-users. These end-users access the server by using Gopher applications of remote computers. Gopher browsers initially display the text-based files. Most of the files along with database are available on Gopher that converts HTTP compatible formats and makes them available on the net.

NOTES



In the preceding screen, the Internet service manager displays the services that are installed on the server to which the Internet service manager is attached.

Finger

Finger service gives information about users, for example, username, person's first name and last name, information about recently logged in and also where they logged in. But the users must enter the required information where they get registration for particular e-mail services. Finger is also used to get a list of users who are currently logged into the host. In fact, the Finger program accepts input as an e-mail address that returns information of user. In some systems, Finger gives the information about the currently logged on users.

World Wide Web (WWW)

WWW provides hypertext access to documents located anywhere on the Internet. It is a very successful distributed information system. It is basically client–server data transfer protocol that communicates via application level protocol. Its structural components are clients–browsers, servers and caches. The Internet and semantic components include hypertext transfer protocol (HTTP), hypertext markup language (HTML) extensible markup language (XML) and uniform resource identifiers (URIs). The clients who get various sites requested to the server via HTTP determine the structure of WWW. Then web pages constructs HTML consists of graphics and sound embedded files. For running the complete system, TCP/IP, DNS networking protocols are required.

NOTES

The reason behind the evolution of Java programming language is to develop distributed application. Distributed application means many CPUs are inter-connected through different network topology so that each CPU can communicate with one another. Java introduced the remote method invocation technique to implement distributed application. The **java.net** package provides classes and methods to develop networking-applications through different network protocols.

A group of computers connected by cable to share information is popularly known as network. A network is a set of computers and peripherals that are physically connected. Networking enables sharing of resources and communication. Java applets can be downloaded from a website. This is one of the main attractions of Java. Networking in Java is possible through the use of **java.net** package. The classes within this package encapsulate the socket model developed by Berkeley software division. The network requires some components, such as:

- Server
- Client
- Peer
- Protocol
- Physical Media
- Physical Devices

Servers provide services to the client. If a server provides application services, then it is treated as an application server.

The **client** accesses services from the server. Peer is a computer that works as a server as well as a client.

Clients

A computer, which requests for some service from another computer, is called a **client**. The one that processes the request is called a **server**. A **server** waits till one of its **clients** makes a request. It can accept multiple connections at a time to the same port number. Multithreading is used to serve multiple users at the same time.

Write the Client side Application

The following example shows this:

Program 1

```
import java.net.*;
import java.io.*;
public class DataClient1
    {
    public static DatagramSocket ds;
    public static int clientport=9999,serverport=10000;
```

NOTES

```
    public static void main(String args[])throws Exception
       {
       byte buffer[]=new byte[2300];
       ds=new DatagramSocket(clientport);
       System.out.println("Client is waiting for server
to send data");
       DatagramPacket                              p=new
DatagramPacket(buffer,buffer.length);
       while(true)
       {
       ds.receive(p);
       String str=new String(p.getData(),0,p.getLength());
       System.out.println(str);
       }
    }
}
```

Steps for Compilation and Execution of the Client Side Application

The various steps are as follows:

(a) Compile client side application: javac DataClient1.java

(b) Execute client side application: java DataClient

Output of the program:

```
D:\net>javac DataClient1.java

D:\net>java DataClient1
Client is waiting for server to send data
```

Servers

Write the Server Side Application

The following example shows this:

Program 2

```
import java.net.*;
import java.io.*;
public class DataServer
    {
    public static DatagramSocket ds;
    public static int clientport=9999,serverport=10000;
    public static void main(String args[])throws Exception
       {
       byte buffer[]=new byte[2300];
       ds=new DatagramSocket(serverport);
```

NOTES

```
        BufferedReader br=new BufferedReader(new
              InputStreamReader(System.in));
        System.out.println("Enter Text");
        InetAddress ia=InetAddress.getByName("localhost");
        while(true)
           {
           String s=br.readLine();
           if((s==null)||s.equalsIgnoreCase("end"))
              {
              break;
              }
        buffer=s.getBytes();
        ds.send(new DatagramPacket(buffer,buffer.
length,ia,
           clientport));
        }
     }
}
```

Steps for Writing, Compilation and Execution of the Server Side Application

The various steps are:

(a) Write the server side application.

(b) Compile it, javac DataServer.java.

(c) Execute it, java DataServer.

Output of the program:

```
D:\net>javac DataServer.java

D:\net>java DataServer
Enter Text
_
```

Communication between Server and Client Application Through Multithreading

This can be seen as follows:

Write the Server Side Application

The following example shows this:

Program 3

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
```

NOTES

```
import java.net.*;
public class AppServer extends Frame implements
ActionListener,Runnable
   {
   Button b1;
   TextField tf;
   TextArea ta;
   ServerSocket ss;
   Socket s;
   PrintWriter pw;
   BufferedReader br;
   Thread th;
   public AppServer()
      {
      Frame f=new Frame("Server Side Chatting");
      f.setLayout(new FlowLayout());
      f.setBackground(Color.orange);
      b1=new Button("Send");
      b1.setBackground(Color.pink);
      b1.addActionListener(this);
      tf=new TextField(15);
      ta=new TextArea(12,20);
      ta.setBackground(Color.cyan);
      f.addWindowListener(new W1());
      f.add(tf);
      f.add(b1);
      f.add(ta);
      try{
         ss=new ServerSocket(12000);
         s=ss.accept();
         br=new BufferedReader(new
          InputStreamReader(s.getInputStream()));
         pw=new PrintWriter(s.getOutputStream(),true);
      }catch(Exception e)
         {
         }
      th=new Thread(this);
      th.setDaemon(true);
      th.start();
      setFont(new Font("Arial",Font.BOLD,20));
```

NOTES

```
        f.setSize(200,200);
        f.setLocation(300,300);
        f.setVisible(true);
        f.validate();
        }
private class W1 extends WindowAdapter
    {
    public void windowClosing(WindowEvent we)
        {
        System.exit(0);
        }
    }
public void actionPerformed(ActionEvent ae)
    {
    pw.println(tf.getText());
    tf.setText("")    ;
    }
public void run()
    {
    while(true)
        {
        try{
        ta.append(br.readLine() +"\n");
        }catch(Exception e)
            {
            }
        }
    }
public static void main(String args[])
    {
    AppServer a=new AppServer();
    }
}
```

Steps for Compilation and Execution of the Server Side Application

The steps are:

(a) Compile server side application, javac AppServer.java.

(b) Execute server side application, java AppServer.

NOTES

Output of the program:



Write the Client Side Application

The following example shows this:

Program 4

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
public class AppClient extends Frame implements
ActionListener,Runnable
    {
    Button b;
    TextField tf;
    TextArea ta;
    Socket s;
    PrintWriter pw;
    BufferedReader br;
    Thread th;
    public AppClient()
       {
       Frame f=new Frame("Client Side Chatting");
       f.setLayout(new FlowLayout());
       f.setBackground(Color.orange);
       b=new Button("Send") ;
       b.addActionListener(this);
       f.addWindowListener(new W1());
       tf=new TextField(15);
       ta=new TextArea(12,20);
       ta.setBackground(Color.cyan);
       f.add(tf);
```

NOTES

```
        f.add(b);
        f.add(ta);
        try{
           s=new Socket(InetAddress.getLocalHost(),12000);
           br=new BufferedReader(new
              InputStreamReader(s.getInputStream()));
           pw=new PrintWriter(s.getOutputStream(),true);
               }catch(Exception e)
           {
           }
        th=new Thread(this);
        th.setDaemon(true);
        th.start();
        setFont(new Font("Arial",Font.BOLD,20));
        f.setSize(200,200);
        f.setVisible(true);
        f.setLocation(100,300);
        f.validate();
        }
private class W1 extends WindowAdapter
    {
    public void windowClosing(WindowEvent we)
        {
        System.exit(0);
        }
    }
public void actionPerformed(ActionEvent ae)
    {
    pw.println(tf.getText())    ;
    tf.setText("")    ;
    }
public void run()
    {
    while(true)
        {
        try{
              ta.append(br.readLine()+"\n")    ;
          }catch(Exception e)    {}
        }
    }
```

```
public static void main(String args[])
    {
    AppClient a1=new AppClient();
    }
}
```

NOTES

Steps for Compilation and Execution of the Client Side Application

The steps are:

(a) Compile the client side application, javac AppClient.java.

(b) Execute the client side application, java AppClient.

1.2.2 Equipments Required for an Internet Connection

Surfing the Internet is quite similar to scuba diving, with regard to the sophisticated equipments deployed to access Internet. Just as we need certain equipments to dive in the deep ocean for scuba diving, we require adequate paraphernalia to successfully plunge into the huge ocean of interconnected computers and networks.

Now-a-days, Internet access necessitates a broadband connection, which is, a high data rate Internet access. The dial-up access deploys a 56K dial up modem, which uses a dedicated telephone line and is limited to the bit rate of less than 56 Kbps. In contrast to this is the broadband technology, which provides more than double the dial up bit rate and that too without intervening with the telephone use. In other words, it means that Internet access and voice call can be carried out simultaneously. The broadband connections are characterized by various minimum bandwidths ranging from 64 Kbps up to 2.0 Mbps. Some standards define the broadband connection as having download data transfer rates equal to or faster than 256 Kbps, whereas others define it as having data transmission speed exceeding 768 Kbps in either downstream or upstream direction. In general, any connection of 256 Kbps or greater comes under broadband Internet.

Certain equipments which are required to access the Internet are as follows: Amongst these, some of them are mandatory and some are optional:

- **Computer:** A computer which is used to browse the Internet may either be a personal computer with Pentium processor or a Macintosh. It should have enough power and memory concomitant with multimedia features. Though 128 MB RAM is sufficient to have access to Internet but 512 MB RAM or more is recommended. Now-a-days, devices like smart phones, mobile phones, Pocket PCs, etc. are also used to browse the Internet.
- **Modem:** It stands for Modulator/Demodulator. This may either be internally built in or externally connected. The modem is a device that converts data in binary code used by the computer, to an analog signal that can be transmitted over the telephone network and vice versa.

NOTES

With the help of telephone lines, millions of computers worldwide are connected with one another, either directly or indirectly. In order to connect with the Internet Service Provider (ISP), these connections require the regular dial up telephone lines or dedicated higher capacity telephone lines like leased lines, ISDN lines, etc.

- **Internet Account with a Service Provider:** An account with a service provider is essential to create a link between the user's computer and the Internet. A service provider, which is popularly referred to as ISP (Internet Service Provider), signifies phone or cable companies that provide last mile connectivity. It may also refer to a cable line from the subscriber's home to his office and also to an exchange for long distance connectivity based on monthly or annual charges.
- **Widely used Current Standard Broadband Technologies:** These technologies are DSLs, that is, Digital Subscriber Line (DSL) and cable modems. However, recent technologies like VDSL and optical fiber connections are also gradually becoming popular in providing Internet access in a much more cost-effective way than copper wire technology. Wi-Fi networks are also used to provide Internet connections. However, these are not served in the areas by cable or ADSL. The WiMAX has been gaining popularity with regard to mobile and stationary broadband access.
- **Internet Browser Software:** It is the software tool which enables a user to browse the Internet with the help of web addresses or URLs. A few of the widely used browsers are Internet Explorer version 7 or 8 (IE), Netscape, Mozilla Firefox, Chrome, AOL, Opera, etc.
- **Anti-Virus Software:** These are used to protect the user from the onslaught of the nasty programs that obtain access to the user's terminal when he is surfing the network or downloading contents from there. Some examples of anti-virus software are Symantec, Norton, McAfee, etc.
- **Email Software:** The email software may be chosen from the Outlook or Outlook Express. Google, Yahoo and Hotmail offer free web-mail for the same.
- **Plug-In Software:** It is considered an add-on to the user terminal. It enables the user to avail services like music, video, multimedia, etc. on the Internet. The most popular plug-in-softwares include Real Audio music player, Macromedia Flash Player, Windows Media Player, Apple Quick Time, Java Virtual Machine, etc.
- **Stereo Speakers, Microphone and Webcam:** These equipments enable the user to play sounds, videos, to conduct Internet telephoning and to send images to other users connected to the Internet.

1.2.3 Home Page

NOTES

Home page is known as the first page of the web page. It is replete with a myriad of hyperlinks on its page. Creation of a home page connotes creating and launching of the website. This is a consequential task which is accomplished by arranging the website hosting, designing and coding of the website, monitoring the functioning of the site and by scrutinizing the website traffic. Creating the website takes into consideration, various factors which are to be implemented on the page. Launching the website is an important operation. This requires information pertaining to name, phone, URL description as well as the domain details. It should be ensured that the website must be kept in the right direction. A comprehensive user's guide that conveys the relevant information of the website, must be provided to the user. This can be done after successfully launching the website. The more accurate the details are, the better the results would be. The task of launching the website can be carried out in the local listing of Google and Yahoo. This optimizes the search engine facilities for your website, which offers moderate list of options, searchable information and the third party data providers, such as SuperPages, YellowPages, CitySearches, etc. These search engines provide a great facility to recite the name of your website. Such search engines also offer a free Jumpstart program in which you can enter you website domain area, avail the 'WAY' (Who You Are) facility, get reviews and list hours etc. There are many factors that determine the success of a website on the net. The following factors should be taken into consideration while creating and launching a Website:

- **Message Board:** It is a type of forum through which visitors of the website interact with the site to enhance its popularity.
- **Search Engine:** This is a valuable retention tool which helps visitors search for relevant information, provided the site is enlisted with a famous search engine.
- **Polls:** This option on the website enables the visitors to vote as per their satisfaction. For instance, the users can assess the performance of the web services by giving their feedback with the help of the feedback option.
- **Guestbooks:** This website facility helps the users contact the organization for which the particular site is created. Through this facility, the website visitors can enter their name, email, comments and suggestions. Once this information reaches the organization, the respective executive of the organization contacts the visitors.
- **Data Entry Forms:** Through this option, the website visitors can place orders and can also provide request information. Date entry forms also facilitate storing of customer service data, which is later entered into a computerized database or spreadsheet by the organization.

NOTES

Customer Record 1

Customer Information

Customer No.
First Name
Last Name
Sex
Address Line 1
Address Line 2
City
State
Post Code
Country

Submit to database or Add another customer record?

Fig. 1.2 *Data Entry Form for Customer Record*

Figure 1.2 depicts the data entry form for customer record that provides various text fields, submit buttons, links to navigate to another page, combo boxes etc. If you click on the link Add another customer record? as specified in the figure, it will provide another set of customer record fields to fill the detail of customer information.

Creation of a Home Page

Creation of a home page requires eight steps which are as follows:

1. **Select and Register a Web Page Domain Name**

   First select a suitable website domain name to monitor the conflict issues of the same. Once a domain name is allotted to an organization or an individual, it can not be further allotted to anyone. The registration of domain name is unique and is carried out by Internet Corporation for Assigned Names and Numbers (ICANN), which is an -accredited domain name registrar, such as abc.com, xyz.com etc. The free website hosting service is also available that can be availed without registering a domain name. The search engine does not provide its services if any website lacks its registered domain name.

2. **Select and Configure a Website Hosting Service**

   The hosting cost of website ranges from $100 to $250 every year. The cost varies from one website to another, depending on the websites' features such as ecommerce facilities, special processing requirements, high traffic volume options, etc. At this stage, web hosting is checked for control over content, security and usage of the site.

NOTES

3. **Design, Code and Test the Website**

   A static website comprises of a single web page. It must have 'index.html' or 'index.htm'. A bare-bone format of the web page, which constitutes the HTML code, is as follows:

```
<HTML>
  <BODY> Hello Web! </BODY>
</HTML>
```

   There are various types of software tools such as Adobe PhotoShop, Microsoft FrontPage etc.

4. **Deploy the Website to the Host**

   At this stage, the file transfer program is uploaded in order to download the website. It also updates the pages between system units and website host computer.

5. **Security and Authentication**

   Before launching a website on the net, it is essential to implement the security and full-proof authentication of the site. The following security methods are required for the same:

   - Login pages should be encrypted.
   - Data validation should be conducted in the server-side.
   - Managing the website with the help of encrypted connections.
   - Website must be connected from secured network.
   - Login credentials must not be shared.
   - Maintaining a Password and key authentication.
   - Use redundancy to protect the website.

   After accomplishing the authentication only SSL is used with **http://URL** scheme is used. The login form POST is encrypted after every login process. The JavaScript is used for validating the web forms. The server side validation safeguards from malicious security crackers but in a limited way. The website must be equipped with encrypted connections because non-encrypted connections make the website susceptible to login or password sniffing or even man-in-middle-attacks.



Fig. 1.3 *Login Screen*

NOTES

A secured connection must be associated with website and also secured proxy server must be used, such as **PuTTY** secure proxy or **OpenSSHproxy**. In order to maintain secure workstation, integrity auditing must be conducted. The server failover and backups should also be deployed as they diminish the possibility of server crashes. Data backup is also important in so far that it safeguards from loosing the client's data.

6. Online Payment mode

Before launching any website, it is essential to set up an online payment mode, which is provided to the website. A website must be equipped with facilities like payment through credit cards or by a third-party such as Paypal, etc.

Other factors that should be considered while launching a website are following:

- A website is launched with the help of File Protocol Program (FTP). It is an economical option. The owner of website must instruct the web designer and the system analysts to implement FTP before the launch of the website.
- The web host firm place on the server must be provided on the site. For example, site owner provides the disk to the web host firm so that they can set up fee for the site.
- It is incumbent upon the owner of the website to remove the 'teething' problem before launching the website. Incoherent or incomplete website can discourage the visitors. For example, if an organization provides e-commerce services, it must ensure that up to date and relevant information is available on its site. The teething problem may lead to problem of set up and layout of the web screen.

A websites is written in HTML and is a collection of linked web pages on a web server which can be electronically accessed. Web server is a machine in which a website is located or hosted. It may be organization owned or Internet Service Provider (ISP) owned. The web pages may be owned by a university, a private company or an individual and are accessible to all people. Most of the websites have their own homepages that facilitate navigation by providing links to explore the details stored therein. The pages of a good website are organized using a common theme.

7. Launching the Website

Launch of the site is carried out after designing and completion of the site. It is essential to finalize the layout and style of the site before launching. It is significant to note that before the launch of the website, its domain should be registered.

NOTES

8. **Promote the Website**

   The information is sent on the web through search engines and their related directories. The promotion scheme must be published on the website at regular intervals. Therefore, this factor must be considered during creation of the website. The optimal way to promote the website is to update the visitors on the specific website with the pertinent information. For example, in case of online air ticket booking systems, any promotional scheme such as shifting the seat arrangement from economic to business class or changing the flight schedules etc. must be updated online to intimate the travellers and the visitors about the same.

It provides a point of entry to a Website with help. It also contains all relevant links of a particular website, so as to enable the user to explore the website for information available therein.

1.2.4 Internet and URL

URL denotes Uniform Resource Locator. It is the address of a document on the World Wide Web. Web browsers enable a person to enter either a known address in the web server or a specific document within that server. Addresses usually begin with http://, ftp://, gopher://, WAIS://, file:// etc. It is not feasible to maintain WWW without using the URLs. These are also used to represent hypermedia links and links to network services within the HTML documents. Any file or service on the Internet can be presented with the help of the URL. The first part of the URL that comes before the two slashes specifies the method of access or protocol being followed for communications between the browser and the web server. The second part coming after two slashes represents the address of the host machine, whose data or services are being sought. The remaining parts signify the names of the files, the port to connect to or the text to search for in a database. All the parts of an address for obtaining a file or service from a host machine in a URL are shown as a single unbroken line with no spaces and the locations of the host machines or websites that run www servers are typically named with a www at the beginning of the network address. The web browsers enable the users to access web services by specifying a URL and connecting to that document or service. Once the user gets connected with the web server, the web browsers select the hypertext in an HTML document and send a request to open a URL. Thus, hyperlinks are used not only to provide other texts and media in the same document but also to facilitate other network services. Web browsers are not simply web clients. They are full-featured FTP, Gopher and telnet clients.

1.3 DOMAIN NAME SYSTEM (DNS)

The Domain Name System (DNS) is a distributed database that provides e-mail routing information. It is used by TCP/IP protocols. These protocols map between IP addresses and hostnames. The domain name extension type is tagged with

NOTES

distributed database because no single website across a network can access all the information. Each site runs a server program for sites and maintains its own database of information. Therefore, collecting information is possible through distributed method that creates a mechanism for clients and servers to communicate with each other.

The domain name extension is defined as the complete address of hosting services provided on the sites. In the beginning, the Internet configuration used numeric IP address, which was a very cumbersome task. To overcome this problem, symbolic host names came into existence. For instance, initially, it was typed as TELNET 10.12.7.14 but nowadays, TELNET MyHost command is issued. With this command, the mapping between machine names and IP addresses has also become centralized and coordinated. A name space is organized in two ways—either hierarchical or flat. It basically maps each address to a unique specified name. The address mappings of host names are maintained by Network Information Centre (NIC) which maintains a single file known as the flat name space. Hierarchical name space involves the nature and name of the organization but in this case, the controlling authority is decentralized. The design of domain name space is hierarchical. It follows inverted tree structure, that is, from the root to the top. It can have up to 128 levels that start from level 0, that is, the root level and may go up to level 127. A label can have a maximum of 63 characters. The root level contains a Null string. Each node in the reversed tree keeps a domain name. A fully qualified domain name contains a sequence of labels separated by dots. The first part defines the nature of an organization, the second part signifies the name of the organization and the third part refers to the department of the organization. (see Figure 1.4). The authority to assign and control the name spaces can be decentralized. For example, myHost.myDept.myDiv.myCorp.com

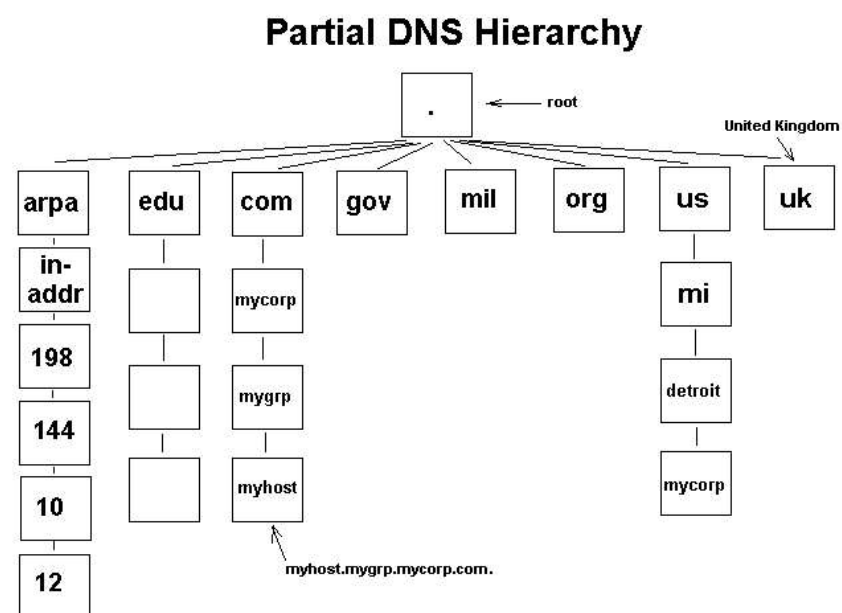


Fig. 1.4 Domain Name Extension Hierarchy

The domain name space is hierarchical in design. The names are defined in an inverted-tree structure with the root at the top. The tree can have 128 levels, that is, level 0 (root) to level 127. Each node in the tree has a label, which is a string with maximum of 63 characters. The root label is a null string. DNS requires that the children of a node have different labels. Each node in the tree has a domain name. A full domain name is a sequence of labels separated by dots. The domain names are always read from the node to the root. Table 1.2 shows the top-level domains that are assigned in the United States.

If the domain name ends in a dot, it shows that the name is not complete. This is known as AND. It is referred to as a Fully Qualified Domain Name (FQDN) or an absolute domain name. For example,

myDept.myDiv.myCorp.com.

If the domain name does not end in a dot, then it means that it is complete. This is called partially qualified domain name (PQDN). For example,

myDept.myDiv.myCorp

The nature of DNS is distributed where symbolic names are grouped into zones of authority. These zones contain a database of symbolic names along with IP addresses. Each zone is a part of sub-tree of hierarchical structure. The names are administered independently within the zones and can be assigned to other zones. The name server includes authority over the zones. Figure 1.5 shows that the domain name can be searched by its extension type.

Fig. 1.5 Searching Domain Name by its Extension Type

Now we have two types of IP addresses in the form of decimal numbers and text for the same host. You know that list of all IP addresses are maintained centrally by ICANN in the form of distributed database directory. There are several distributed servers, which maintain this list of IP addresses. The reasons behind the distributed server are very logical and simple. It helps in disaster management and in diverting the load of traffic in the form of requests from clients to other DNS servers located at different sites. DNS server maintains database in both the textual and the decimal notation form. For example, DNS server maintains the address of the google site as www.google.com and 216.23.9.53.99. In this manner, DNS is used to provide host-to-IP address mapping of remote hosts to the local hosts and vice versa. It is now conspicuous that the DNS maintains a distributed database to map between hostnames and IP addresses. Whenever a client requests a service from a site, both the sites run DNS protocol to access the distributed database which is nothing but Domain Name Systems. Therefore, the DNS provides the

protocol, which allows the clients and servers to communicate with each other. DNS also helps the system use a resolver, which resolves the host name to that IP address, which can be understood by the server.

NOTES

Now let us talk about how DNS is able to quickly translate the text of the IP addresses, from a directory of billions of such addresses and that too within a fraction of seconds.. This could be made possible by using Domain concepts, which use hierarchical arrangements of text addresses translation.

As illustrated in the Figure 1.6 you can see that at the top level is the root server, which has a null label. Below this is another level domain, namely com, edu, int which are grouped together. Below this different sub domains or groups have been created. Table 1 depicts some commonly appearing domain names with their respective sites. The DNS can accommodate almost all kinds of organizations by allowing each group to choose between geographical or organizational naming hierarchies.

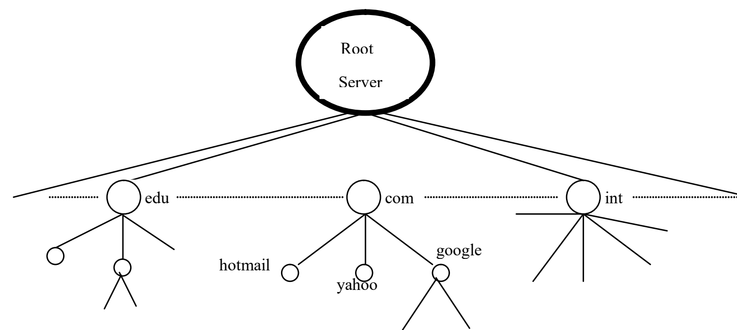


Fig. 1.6 *DNS Hierarchy*

Table 1.2 *Internet Domains*

Domain	Indicative Site
Com	Commercial institute
Edu	Educational institute
Org	Non-profit organization
Net	Network service provider
Gov	Government department
Mil	Military
Biz	Business
Country code	For example, in for India, us for USA, au for Australia, jp for Japan and so on

As we know that the servers maintaining addresses are disseminated and have locations throughout the world. Now the questions arises that how text addresses are organized in hierarchical arrangement. Now consider the Figure 1.6 and Table 1.2 given above. The hierarchy is represented through zones and

NOTES

each zone depicts a hierarchy of one or more nodes without any overlapping. Each zone is represented by a server and undoubtedly with one backup server. The Root server, as shown in Figure 1.6, is the only one which is just indicative. There may be several root servers at several locations in the world. Each root is aware of the location of each DNS server of a specific domain.

The process is now very simple to understand. When you need to connect with a particular site, you first send your request to your local host. If your local host provides the translation, your request is completed. If not, your local host then sends your request one level above in the hierarchy. If the server at one level above is able to handle the same, you get your intended website at your desktop through your local server. If not, then the server at one level above your local server either sends your request again to another server or informs your local server that your request has failed and eventually gives you the address of another server to process your request. This process continues until a server, which knows the address, is found. Otherwise, the request is filtered up to the root server. Depending upon the domain address, the root server forwards the request to one of the domain servers represented at the next level of hierarchy. This process continues until the information of text address is returned to the Root server and then back to your local server.

1.4 E-MAIL

Electronic mail is one of the most popular network services. The use of e-mail is considered the foremost reason behind the popularity of Internet. The proliferation of cyber cafés can be attributed to e-mail or World Wide Web. E-mail provides an efficient and fast means of communication with relatives, friends or colleagues throughout the world. With the help of email, one can not only communicate with myriad people at a time but can also receive and send files and other information within a fraction of seconds. The biggest advantage of email is that the intended receiver of the message does not require to be present at their desktop at the time of receiving of the message.

Definition

The term email connotes the basic communication facility provided by the Internet to its users to send and receive messages in any part of the world. It is considered one of the most popular applications of the Internet and is accounted for 90% of net traffic. Email facilitates sending of messages in the form of a text, audio and video or even a combination of these types. When a message is sent from the source user, it reaches the recipient's mail box. The email message received by the recipient can be opened, discarded, edited, saved, responded back to or can even be forwarded to some other recipient. Email messages are delivered instantly after the transmission. An email can be sent by connecting to the network from any location. An Internet connection usually requires a telephone line, a modem and a

computer. Wireless connections have also become popular means of getting connected to the Internet. This job is accomplished by the Simple Mail Transfer Protocol (SMTP) running over TCP/IP.

NOTES

Uses of Email

Email provides several features that are useful in day-to-day life. It is an efficient and cost-effective way of communication across the world. With the help of email, one can send common letters or circulars to several recipients. The email messages are delivered instantly, even if they are sent to remote locations worldwide. Thus, it saves time as well as money. Whereas the postal messages are time consuming. Email also provides an address book facility which keeps a record of the email addresses. This saves the user from the predicament of remembering the addresses of the recipients. In addition, a lot of time, energy and money is saved as the user creates a mailing list with a group name, so that a letter or a circular can be transmitted by just typing the name of the particular group. Another advantage of using email is that provided the email address typed in is correct, it enables the sender to know immediately whether the message has been delivered to the recipient,. In case the message is not delivered, the sender will receive a return email message to inform him about the failure of the particular message. Email goes beyond all time zones and barriers.

Email also provides the user with a facility of attachment which allows the user to attach any file created in any application such as word processors, spread sheet or power point presentations. For example, if the total amount of outstanding against a client is computed in a spreadsheet, the client may be informed through a letter in email along with an attachment showing his outstanding amount in the spreadsheet. The primary advantages of email can be summarized in the following:

- It conducts paperless communication of messages quickly.
- It ensures simultaneous transmission of messages to several users. The messages may comprise of pictures, video, film clips, text, animation or even a combination of them. Voice and audio messages can also be transmitted this way.
- The email messages can also be printed, prioritized, forwarded and stored.
- Public bulletin boards can be created in which every member of the organization can post and view messages. This can also be accomplished in the case of shared text messages and application files used widely across computer platforms.
- It allows delivery and receiving of faxes and meetings can also be scheduled through email.

NOTES

1.4.1 Opening and E-Mail Account

Opening an E-mail account is not an issue. Now-a-days, all subscribers get facility to open an email account free of cost. A number of web services like Google, Hotmail, Yahoo, etc. are readily available to register a user to open an email account and access it from anywhere in the world. However, to avail this facility, the user should have access to a computer and an Internet connection. In addition to these web services, organizations or ISPs also provide web interfaces to enable the users to open their email accounts, though by charging them. In this case, the organization or ISP possesses the personal record of the users and based on their personal records and their relationship with them, they open their email accounts and equip them with an email address. The email addresses comprise of email ids meant for individual users, which could be their first name or a combination of their name and surname or their date of birth, etc. along with the URL of the organization. For example, in sanjay0203@teraclean.com, sanjay 0203 signifies email id consisting of the name and birth date and month, whereas teraclean.com indicates the URL of the organization.

In case of universally available web services like Google, Hotmail or Yahoo, the user needs to open the website of the respective Web service by typing its corresponding URL in the Web browser. For example, if the user wants to open an account in the Google Web service, then he needs to key the Web address of Google, that is, www.google.com. Once the Website of Google opens, the user needs to click on the Gmail service of the Google. The Gmail interface provides the facility for opening of a new account, for which it provides a registration form to be filled up by the user. In accordance with the procedure, the user mentions his personal information, email id and password in the form. Thereafter, he gets registered and obtains an email address. This process of creating an email account is described as follows:

Type the URL "http://www.google.com" in the address bar of a Web browser such as an Internet Explorer, to visit the Google homepage as shown in Figure 1.7.

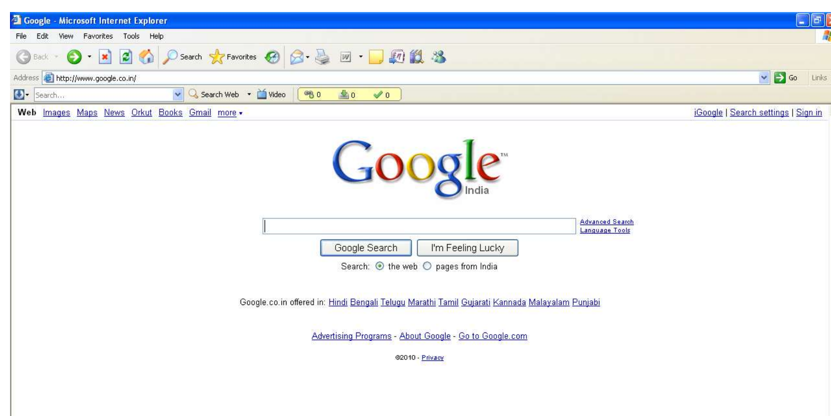


Fig. 1.7 *Google Homepage*

NOTES

The page displayed shows an icon namely Gmail as shown in Figure 1.7. Once you click on the Gmail icon, it navigates you to another webpage as shown in Figure 1.8. If you have an existing account with Gmail, you can type in your email id and your password to log on to your account. If you are accessing the Gmail for the first time, then you need to create an account for yourself. The procedure for the same is as follows:

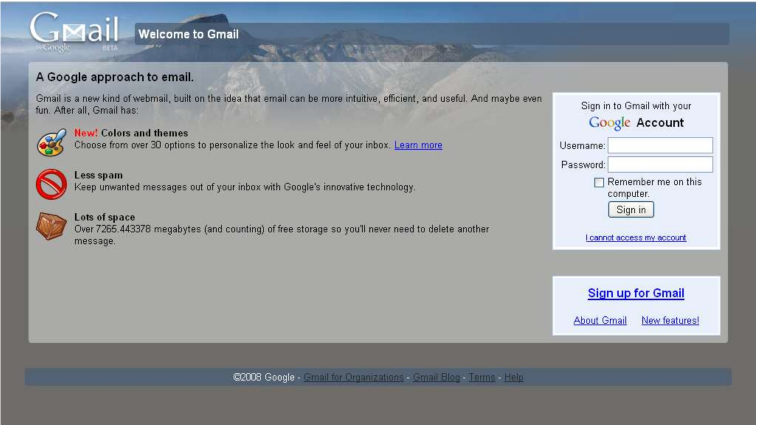


Fig. 1.8 *Sign-in Page*

Click on the "Sign up" icon as illustrated in Figure 1.8. This will lead you to another webpage that contains the registration form as shown in Figure 1.9. Now you are required to fill the form that asks for your personal details along with your user ID and password to open a new email account for you.

Gmail by Google

Create a Google Account - Gmail

Create an Account

Your Google Account gives you access to Gmail and other Google services. If you already have a Google Account, you can sign in here.

Get started with Gmail

First name:

Last name:

Desired Login Name: @gmail.com

Examples: JSmith, John.Smith

check availability!

Choose a password: Password strength:

Minimum of 8 characters in length.

Re-enter password:

Fig. 1.9 *Registration Page*

Once you are through with the registration process, after accepting the terms and conditions, you become a member and thereafter you are able to use your email account to send and receive emails. Now all you need is to remember your user ID and your password for future use of your email account. In other words, it means that whenever you need to log on to your newly created account, you need to simply type your user ID and your password.

1.4.2 Reading and Writing E-Mail

Email is extensively used by people across the world. The procedure of reading and writing an email is not a very sophisticated one. The steps involved are as follows:

Reading an E-Mail Message

The email account can be accessed at anytime and from anywhere by logging on to the particular email account, as mentioned earlier. To read or write an email, you need to perform the following steps:

1. Type the URL "http://www.google.com" in the address bar of a Web browser.
2. Enter your user ID and the password as shown in Figure 1.10.



Fig. 1.10 *Sign-In Page*

Once you have signed in successfully, you can access your email account as shown in Figure 1.11.

NOTES

Fig. 1.11 *E-Mail Account*

Clicking on the Inbox icon lets you open your Inbox. The Inbox folder contains all your previous email messages and also enables you to read the new ones. You also have an option of deleting the previous messages or transferring them to some other folders also. An email message in the Inbox can be read by clicking on the email subject or any other clickable item therein. This displays the contents of the message to be read and allows you to take appropriate action accordingly. Some email messages are delivered along with attachments. Attachments may comprise of textual messages, graphics, pictures, videos, sounds or a combination of these types.

The email message depicts an attachment button within the message itself, which on being clicked enables you to either open the attachment in relevant applications or save it on your computer to be opened separately.

Writing an E-Mail Message

AS mentioned earlier, email account can be accessed by logging on to email account. To write an email, you need to perform the following steps:

The Compose option on the left hand side of the screen enables you to write an email message. Attachments can also be appended along with the email messages wherever they are required. On selection of the compose option, a screen as shown in Figure 6 will appear. The following steps are to be followed for writing and sending an email message:

1. **To:** It is a field in which the valid email address of the recipient like User ID@domain.com is typed in, so that the message can be delivered correctly. In case of multiple recipients, email address of each recipient is typed in the same box separated by comas.

2. **Cc:** It signifies the email address/(s) of the recipient/(s) to whom a carbon copy of the message is to be transmitted. The recipient/(s) specified in To field also receives the email address/(s) of the recipients in their messages indicating that email address/(s) in the Cc field also receive/(s) the same message.
3. **Bcc:** It denotes the email address/(s) of the recipient/(s) to whom a copy of the message is transmitted. However, in this case, the recipient/(s) in both To and Cc field remain oblivious of the other email addresses, to which the message is sent. Bcc stands for blind carbon copy.
4. **Subject:** This box enables the sender to write the subject of the message, so that recipient/(s) on receiving the message, could have a clear idea of what the email message is about.
5. **Message Box:** It is the field in which you type your message which is to be transmitted.

An attachment can also be appended to the email message before sending it. There exists an Attachment button within the compose mail box. On clicking on the Attachment button, you are asked to provide the location of the desired file to be attached. You then click on the Browse button which enables you to select the desired file from your computer. Finally, clicking on the Attach or OK button attaches the document along with your email message.

Your message with or without attachment is now ready to be transmitted. Now you need to follow the following steps:

1. If you want to postpone transmitting of your message, you have another option called Draft in which you can save your message to be transmitted later. The message saved in the Draft can also be modified before transmission. The Draft webpage provides you a Send button. On clicking on it, your message is transmitted and a copy of the message is saved in your Sent mailbox, provided the send and save option has been set.
2. If you do not want to postpone the transmission of your message, then just click on the Send button. On clicking on it, your message will be transmitted and a copy of the message will be saved in your Sent mailbox.

Check Your Progress

1. What is Internetworking?
2. Define the term ARPAnet.
3. What is Home page?
4. Explain the term Domain Name System (DNS).
5. What is E-mail? What is the advantage of e-mail?

1.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. Internetworking is a scheme for interconnecting multiple networks of dissimilar technologies. To interconnect multiple networks of dissimilar technologies use both additional hardware and software. This additional hardware is positioned between networks and software on each attached computer. This system of interconnected networks is called an Internetwork or an Internet.
2. ARPAnet was basically a network based on leased lines connected by special switching nodes, known as Internet Message Processors (IMP). Many researchers were involved in TCP/IP research by 1979. This motivated DARPA to form an informal committee to coordinate and guide the design of the communication protocols and architecture. The committee was called the Internet Control and Configuration Board (ICCB).
3. Home page is known as the first page of the web page. It is replete with a myriad of hyperlinks on its page. Creation of a home page connotes creating and launching of the website.
4. The Domain Name System (DNS) is a distributed database that provides e-mail routing information. It is used by TCP/IP protocols. These protocols map between IP addresses and hostnames. The domain name extension type is tagged with distributed database because no single website across a network can access all the information. Each site runs a server program for sites and maintains its own database of information. Therefore, collecting information is possible through distributed method that creates a mechanism for clients and servers to communicate with each other.
5. Electronic mail is one of the most popular network services. The use of e-mail is considered the foremost reason behind the popularity of Internet. The proliferation of cyber cafés can be attributed to e-mail or World Wide Web. E-mail provides an efficient and fast means of communication with relatives, friends or colleagues throughout the world. With the help of email, one can not only communicate with myriad people at a time but can also receive and send files and other information within a fraction of seconds. The biggest advantage of e-mail is that the intended receiver of the message does not require to be present at their desktop at the time of receiving of the message.

1.6 SUMMARY

- The Internet, World Wide Web and Information Super Highway are terms which have the lives of millions of people all over the world.

- The widespread impact of Internet across the globe could not be possible without the development of Transmission Control Protocol/Internet Protocol (TCP/IP). This protocol suite is developed specifically for the Internet.
- The availability of different operating systems, hardware platforms and the geographical dispersion of computing resources necessitated the need of networking in such a manner that computers of all sizes could communicate with each other, regardless of the vendor, the operating system, the hardware platform, or geographical proximity.
- Internetworking is a scheme for interconnecting multiple networks of dissimilar technologies. To interconnect multiple networks of dissimilar technologies use both additional hardware and software. This additional hardware is positioned between networks and software on each attached computer. This system of interconnected networks is called an Internetwork or an Internet.
- ARPAnet was basically a network based on leased lines connected by special switching nodes, known as Internet Message Processors (IMP). Many researchers were involved in TCP/IP research by 1979. This motivated DARPA to form an informal committee to coordinate and guide the design of the communication protocols and architecture. The committee was called the Internet Control and Configuration Board (ICCB).
- Regional networks connect, for example, universities and colleges. ERNET (Education and Research Network) is an example in the Indian context.
- Commercial networks provide access to the backbones to subscribers, and networks owned by commercial organizations for internal use and also have connections to the Internet. Mainly, Internet Service Providers come into this category.
- The Gopher protocol supports client–server software that searches files on the Internet. A Gopher client is required for validating and testing of Gopher publishing service.
- World Wide Web (WWW) provides hypertext access to documents located anywhere on the Internet. It is a very successful distributed information system. It is basically client–server data transfer protocol that communicates via application level protocol. Its structural components are clients–browsers, servers and caches.
- Home page is known as the first page of the web page. It is replete with a myriad of hyperlinks on its page. Creation of a home page connotes creating and launching of the website.
- The Domain Name System (DNS) is a distributed database that provides e-mail routing information. It is used by TCP/IP protocols. These protocols map between IP addresses and hostnames.

NOTES

- The domain name extension type is tagged with distributed database because no single website across a network can access all the information. Each site runs a server program for sites and maintains its own database of information.
- Depending upon the domain address, the root server forwards the request to one of the domain servers represented at the next level of hierarchy. This process continues until the information of text address is returned to the Root server and then back to your local server.
- Electronic mail is one of the most popular network services. The use of e-mail is considered the foremost reason behind the popularity of Internet. The proliferation of cyber cafés can be attributed to e-mail or World Wide Web.
- E-mail provides an efficient and fast means of communication with relatives, friends or colleagues throughout the world. With the help of email, one can not only communicate with myriad people at a time but can also receive and send files and other information within a fraction of seconds.
- The biggest advantage of email is that the intended receiver of the message does not require to be present at their desktop at the time of receiving of the message.
- The term e-mail connotes the basic communication facility provided by the Internet to its users to send and receive messages in any part of the world. It is considered one of the most popular applications of the Internet and is accounted for 90% of net traffic.
- E-mail facilitates sending of messages in the form of a text, audio and video or even a combination of these types.

1.7 KEY WORDS

- **Internetworking:** Internetworking is a scheme for interconnecting multiple networks of dissimilar technologies. To interconnect multiple networks of dissimilar technologies use both additional hardware and software, this additional hardware is positioned between networks and software on each attached computer.
- **ARPAnet:** ARPAnet was basically a network based on leased lines connected by special switching nodes, known as Internet Message Processors (IMP).
- **Regional networks:** These connect, for example, universities and colleges. ERNET (Education and Research Network) is an example in the Indian context.

- **Commercial networks:** They provide access to the backbones to subscribers, and networks owned by commercial organizations for internal use and also have connections to the Internet. Mainly, Internet Service Providers come into this category.
- **Home page:** Home page is known as the first page of the web page. It is replete with a myriad of hyperlinks on its page. Creation of a home page connotes creating and launching of the website.
- **Domain Name System (DNS):** The Domain Name System (DNS) is a distributed database that provides e-mail routing information. It is used by TCP/IP protocols. These protocols map between IP addresses and hostnames.

1.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

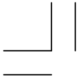
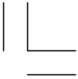
1. Define the term Internetworking.
2. What is Home page?
3. Explain the importance of Domain Name System (DNS).
4. Explain why E-mail is considered an important application.

Long-Answer Questions

1. Discuss the basic Internet concepts with the help of appropriate examples.
2. Briefly explain the concept of Internet and Internetworking giving appropriate examples.
3. Differentiate between client and server applications with the help of programs.
4. Explain the equipment and software required for connecting to the Internet.
5. Briefly explain the structure and significance of Domain Name System (DNS) for Internet.
6. What is E-mail? What are its uses? Give the steps for opening, reading and writing an E-mail.

1.9 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

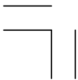
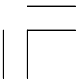
NOTES

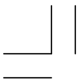
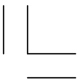
Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

UNIT 2 THE WORLD WIDE WEB

NOTES

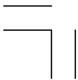
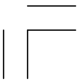
Structure

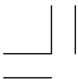
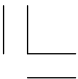
2.0 INTRODUCTION

The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs), such as https://example.com/, which may be interlinked by hypertext, and are accessible over the Internet. The resources of the Web are transferred via the HyperText Transfer Protocol (HTTP) and may be accessed by users by a software application called a 'Web Browser' and are published by a software application called a 'Web Server'.

English engineer and computer scientist Sir Timothy John Berners-Lee invented the World Wide Web (WWW) in 1989. He wrote the first web browser in 1990 while employed at CERN near Geneva, Switzerland. The browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and then to the general public in August 1991. The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet.

Web resources may be any type of downloaded media, but web pages are hypertext media that have been formatted in HyperText Markup Language (HTML). Such formatting allows for embedded hyperlinks that contain URLs and permit users to navigate to other web resources. In addition to text, web pages may contain references to images, video, audio, and software components which are displayed in the user's web browser as coherent pages of multimedia content. A web browser (commonly referred to as a browser) is a software user agent for accessing information on the World Wide Web. To connect to a website's server and display its pages, a user needs to have a web browser program. This is the program that the user runs to download, format and display a web page on the user's computer. A web search engine or the Internet search engine is a software

NOTES

system that is designed to carry out web search (Internet search), which means to search the World Wide Web in a systematic way for particular information specified in a web search query. The information may be a mix of web pages, images, videos, infographics, articles, research papers and other types of files. Some search engines also mine data available in databases or open directories.

In this unit, you will study about the World Wide Web, the Internet search engines, web browsers, chatting and conferencing on the Internet.

2.1 OBJECTIVES

After going through this unit, you will be able to:

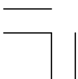
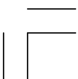
- Explain the significance of the World Wide Web
- Define the importance of the Internet search engines
- Discuss the significance of the web browsers
- Elaborate on the concept chatting and conferencing on the Internet

2.2 THE WORLD WIDE WEB

The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs), such as https://example.com/, which may be interlinked by hypertext, and are accessible over the Internet. The resources of the Web are transferred via the HyperText Transfer Protocol (HTTP) and may be accessed by users by a software application called a 'Web Browser' and are published by a software application called a 'Web Server'.

English engineer and computer scientist Sir Timothy John Berners-Lee invented the World Wide Web (WWW) in 1989. He wrote the first web browser in 1990 while employed at CERN near Geneva, Switzerland. The browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and then to the general public in August 1991. The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet.

You have already studied at length about the World Wide Web (WWW) which is considered a major reason behind the popularity of Internet. WWW refers to a system of information and communications through which the users can access hypermedia information on servers. It is treasure trove of boundless information, in which all items have a reference through which they can be retrieved. In other words, any kind of information on any topic is readily available on the net. WWW comprises of a collection of websites which are publicly accessible. A website usually contains multiple pages which are replete with different types of information about different topics. Following are the constituents of a website:

NOTES

- **Home Page:** This page tells the visitors across net about what is that. It also tells the visitors across net about what is that. The home page also locates the relevant sites on the net. It also provides detailed information about its service domain.

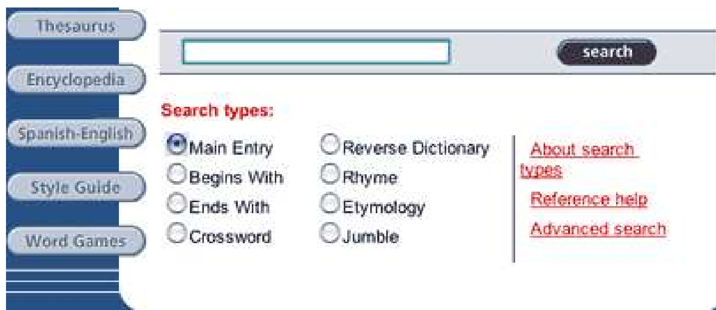


***Fig. 2.1** Home Page Screen*

The Figure 2.1 shows the Home Page of the site that provides various buttons to search the types. The website depicted here is meant for online searching of thesaurus, encyclopedia, style guide, word games and Spanish-English dictionary.

The web service provider helps establish a home page on the net. It is also known as electronic description of the organizations and its products and services. In real life, it is similar to a brochure or a catalog. An attractive home page grabs the attention of the visitors and that is why its cost per page is higher as compared to regular web pages. A home page can be setup with the help of Integrated Services Digital Network (ISDN) line as it generates HTML which represents the graphical interface.

For example, if you type www.google.com, it brings you the Google home page which is illustrated in Figure 2.2.

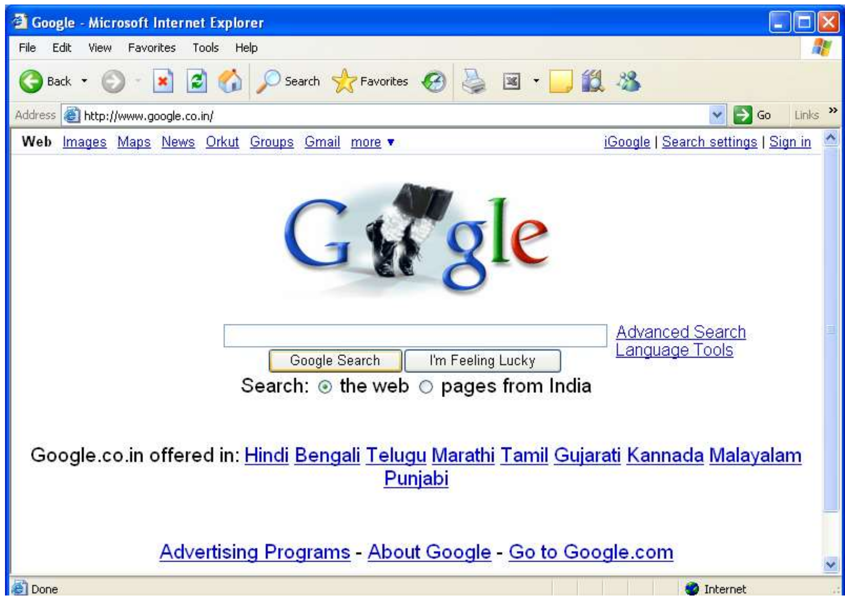


***Fig. 2.2** Google Home Page*

NOTES

- **About page:** This page provides the visitors with a detailed information of the launched website. For example, an online air ticket booking system gives a detailed description of its services for an air reservation domain.
- **Press page:** This page on the site helps people interact with media. It also encourages the visitors to include the address, e-mail and phone/mobile numbers. It also publishes news on its website on a daily basis. Before launching any site, it is essential to check whether the press page, contact page and other attractive features are available on the website.
- **Contact page:** The contact page provides the visitors with the city map, email address, phone numbers and other relevant information, so that the visitors can personally contact the organization or the service providers.

2.2.1 WEB PAGE

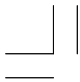
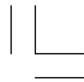
The WWW is a subset of the Internet and comprises of a huge collection of documents stored in computers across the world. The web encompasses special sites called websites along the Internet, that support web browsing. By clicking on the links that appear on the webpage, one can navigate from one place to another. Hence, webpage can be defined as **a** single hypertext document written in HyperText Markup Language (HTML) and described in HTML basics. A webpage normally incorporates the basic information and links to navigate in the websites to which it belongs. Documents in the World Wide Web are classified into three types, namely static, dynamic and active documents.

Static Webpage

These are fixed content documents which perpetually provide the same information in response to all download requests from all web users. Static documents are stored in a web server to be accessed by the web client. The web client, on requesting for a web page, gets a copy of the same. The contents of such files are not subject to modification on part of the web user as the web user does not have right to alter them. However, the web pages can be modified in the server per say. Thus, the static web pages display the same information to all the web users and provide hypertext links to perform navigation through static documents. Their biggest advantage is that they are cache friendly. This enables the web pages to display one copy of the same web page to many people simultaneously. However, it becomes difficult to maintain web pages in case of large sites as they demand consistency and updation.

Dynamic Webpage

These web pages provide interactive web navigation and help modify the content like text, images, form fields, etc. on a web page, depending on different contexts or conditions. The dynamic web pages make use of two types of inter-activities, which are enlisted in the following:

NOTES

- **Client side scripting -** It is used to modify interface behaviours within a specific web page. This modification is based on the mouse or keyboard actions and is conducted at specified time intervals. The dynamic behaviour takes place within the presentation. The presentation technologies like JavaScript or ActionScript for dynamic HTML (DHTML) and Flash for media types of the presentation are used. The client side scripting also facilitates the use of remote scripting in which the DHTML page requests for additional information from the server. The content is generated on the web client's machine in which the web browser retrieves a page from the server and processes the code embedded in the web page, so that the contents of the retrieved page can be displayed to the web user. Sometimes, the web browsers do not support the language and the commands of the scripting language, in the client-side dynamic pages.
- **Server side scripting -** It is used to modify the requested web page source amongst pages to either adjust the sequence or reload the web pages delivered to the browser. Server responses are based on certain conditions like data in a posted HTML form, parameters in the URL, the type of browser being used, the passage of time or a database or server state. Server side scripting dynamic web pages are designed with the help of server-side languages like PHP, Perl, ASP, JSP, etc.

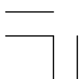
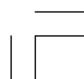
Both the techniques may be used simultaneously to develop the dynamic web pages. The advantages of dynamic web pages are that these facilitate easy update of the web pages and faster web page loading. In the dynamic web pages, the content and the design are located separately, thereby allowing frequent modifications to the web pages including the text and image updates.

Active Documents

The programs that run at the client side are known as the active documents. Whenever a web client requests for an active document, the web server provides a copy of the same in the form of byte code. The document is now ready to be run at the web client machine. As the active document is served in the binary form, compression and decompression can be applied at the server and the client side to reduce the bandwidth requirement and throughput.

2.3 THE INTERNET SEARCH ENGINES

Search engines are the software that enable searching of the content available on Internet. A search engine is an information retrieval system which is used to access and retrieve information stored in WWW or a computer system attached to the Internet. Search engines also help minimize the time required to find the relevant information on the computer system. The computer system could be a standalone system or it could also be attached to the Internet. The search engines are popular

NOTES

amongst people as web search engines help explore information on the World Wide Web.

Search engines are the interface to a group of contents, which allow the users to type in the keywords, so that the engine can find several matching contents to the corresponding keywords out of millions of web pages. The keywords provided by the user are referred to as a search query. Several styles of search query syntax are used by the net users. Search query differs for different types of search engines, that is, some search engines enable users to enter two or three words separated by space, whereas others may require users to provide entire documents, pictures, sounds, and various forms of languages. Some search engines attempt to enhance the search queries to provide a quality set of items through a process known as query expansion.

http://en.wikipedia.org/wiki/Image:Search-engine-diagram-en.svg

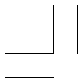
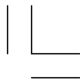
Index-Based Search Engine

In case of such engines, the list of items to meet the criteria specified by the query, is typically sorted or indexed. Indexing the contents by relevance, that is, from the highest to lowest, minimizes the time needed to explore the desired information. Some search engines use probabilistic approach to rank the contents, depending on measures of similarity, popularity or authority. Boolean search engines typically provide contents which match exactly irrespective of the order in which the keywords are typed. However, the term boolean search engine may also allude to the use of boolean-style syntax. Thus, in order to provide a set of matching contents that are based on some criteria, the search engine will collect metadata concerning the group of contents under consideration, through a process called indexing. The advantage of indexing is that it calls for a smaller amount of computer storage.

Types and Characteristics

Some of the popular search engines with their types and characteristics are following:

- Alta Vista – It is a crawler type of search engine which comes up with results based on how many times the search words appear in the text. It searches the complete text.
- Excite – It is also a crawler type of search engine and it makes use of meta tags.
- Google – It is the most widely used search engine. It is also a crawler type and its results are based on the number of times, the other sites link to the ranked site.
- Yahoo – This is also a crawler type search engine and its functions are similar to that of Google.

2.4 WEB BROWSERS

NOTES

A browser is a software, which your computer uses to view WWW documents and access the Internet. The browser program residing in your computer provides you with the facilities like text formatting, hypertext links, images, sounds, motion and other features. Internet Explorer and Netscape are considered the most widely used browsers. Browsers have sub programs called plug-ins to handle the documents found on the Web. It may also have other plug-ins stored elsewhere in the computer.

Web browsers are used to interpret special hypertext pages consisting of the HyperText Markup Language (HTML) and JavaScript so that they can be displayed in the given format. Some of the widely used web browsers are Internet Explorer, Netscape Navigator, Mozilla Firefox, Google Chrome, Lynx, Opera, Apple's Safari, etc.

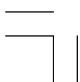
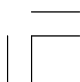
Components of a Web Browser (Browser Architecture)

A Web browser comprises of three parts. These are controllers, client programs and interpreters.

- **Controller:** The controller obtains input from the keyboard or the mouse to access web pages with the help of a client program. After accessing the web pages, the controller uses one of the interpreters to display the web pages on the host screen.
- **Client Programs:** These are used to establish TCP sessions with the web server or the proxy server. To accomplish this task, the client programs make use of HTTP, FTP, Gopher or Telnet.
- **Interpreters:** These are used to display the web pages on the web user's screen. The interpreters which are used to translate web pages on the client's screen are HTML, CGI and JAVA. Such interpreters depend on the type of document. The HTML, which is a markup language and which allows the browser to change the format of the web pages, is used for scripting web pages. The HTML also helps store instructions along with the text, so that any browser can read the instructions and format the text according to the respective host machine.

2.5 CHATTING AND CONFERENCING ON THE INTERNET

Online chat may refer to any kind of communication over the Internet that offers a real-time transmission of text messages from sender to receiver. Chat messages are generally short in order to enable other participants to respond quickly. Thereby,

NOTES

a feeling similar to a spoken conversation is created, which distinguishes chatting from other text-based online communication forms, such as the Internet forums and e-mail. Online chat may address point-to-point communications as well as multicast communications from one sender to many receivers and voice and video chat, or may be a feature of a web conferencing service.

Online chat may be primarily defined as any direct text-based or video-based (webcams), one-on-one chat or one-to-many group chat (formally also known as synchronous conferencing), using tools, such as instant messengers, Internet Relay Chat (IRC), talkers and possibly Multi-User Dungeons (MUDs). The expression online chat comes from the word chat which means 'Informal Conversation'. Online chat includes web-based applications that allow communication – often directly addressed, but anonymous between users in a multi-user environment. Web conferencing is a more specific online service, i.e., often sold as a service, hosted on a web server controlled by the vendor.

The first online chat system was called Talkomatic, created by Doug Brown and David R. Woolley in 1973 on the PLATO System at the University of Illinois. It offered several channels, each of which could accommodate up to five people, with messages appearing on all users' screens character-by-character as they were typed. Talkomatic was very popular among PLATO users into the mid-1980s. In 2014, Brown and Woolley released a web-based version of Talkomatic.

The first online system to use the actual command 'Chat' was created for The Source in 1979 by Tom Walker and Fritz Thane of Dialcom, Inc. Other chat platforms flourished during the 1980s.

Conferencing on the Internet

Conferencing brings together groups of people to share their experiences, information, and expertise. Traditional conferences have required that people share the same physical space and time. With the advent of technology, and telecommunications, conferences no longer require a shared physical space, but still require a coordinated time for participants to meet. The Internet, and other technologies, support conferencing across spatial boundaries by connecting people electronically. There is a variety of Internet conferencing tools, including video cameras, whiteboard software, groupware tools, and Web conferencing products.

Videoconferencing: Video represents real-time moving images, usually at 30 frames per second (fps) or slower, broadcast and received over the Internet. Video is a powerful medium for human communication, since it supports visual and audio signals for social interaction. Video also requires a large bandwidth for transmission and reception, since each second of video represents between 10 and 30 frames.

NOTES

Shared Whiteboard: In the today's business world, most conference rooms have a whiteboard, similar to a school blackboard but with coloured markers instead of chalk for drawing and writing. Shared whiteboards support collaborative writing or drawing on the Internet by sending and receiving the contents of the whiteboard or drawing window to everyone who participates in the conference.

Bulletin Board Systems (or BBS): An online bulletin board is a shared meeting place for people on the Internet. Setting up a BBS involves a dedicated computer as a server, installation of the essential software, establishing users, and operating the BBS.

Chat Windows or Rooms: Chat windows or rooms permit people to share the data electronically what they type. The Internet Relay Chat (IRC) is the original interactive chat feature, and many Web browser programs support IRC.

Web Tours: It is the latest conferencing tool which supports a group of people, all browsing the Web, with a single person acting as the tour guide. The responsibility of the tour guide is to take all online people browsing the Web to remarkable and relevant Websites.

Groupware or Computer-Supported Cooperative Work (CSCW) Tools: Groupware tools support people working together using computer technology. Groupware often includes e-mail, shared scheduling, shared whiteboard, and chat areas, and collaborative writing and design.

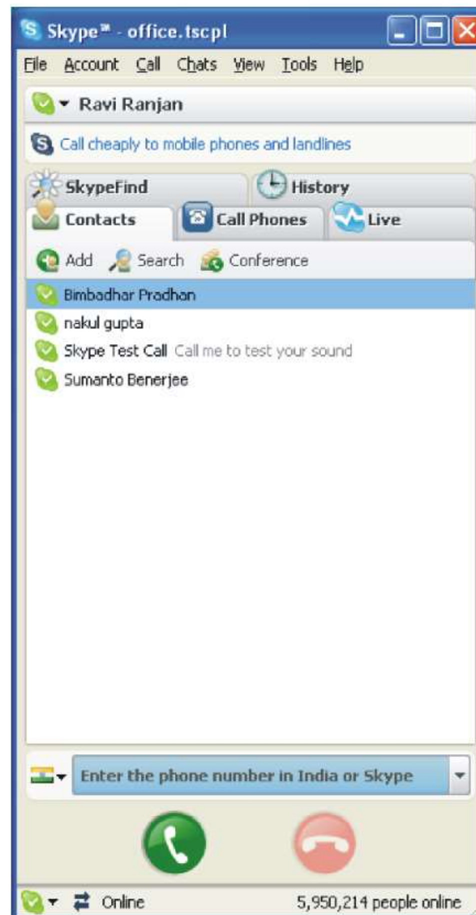
Internet Phone SKYPE

Skype is a piece of software that lets you talk over the Internet to anyone, anywhere in the world for free.

Features

- Free Skype-to-Skype calls can be made to anyone, anywhere across the globe
- Calls can be made to mobiles and other ordinary phones at rates that are cheaper than normal landline or mobile costs.
- Through video calls, you can see the persons you talk to
- SMS (short message service) can be sent to anyone, even when on the move, and the call-forwarding option can be activated to enable them to contact you.
- Group-chats that can include up to 100 people or conference calls including up to nine others can be held
- The Web can be searched with the Google Toolbar (optional install)
- Downloads are free

NOTES



System Requirements

- A PC with Windows 2000 or XP (Windows 2000 users must have DirectX 9.0 to make video calls)
- An Internet connection (preferably broadband; GPRS is not supported for voice calls, and results may vary depending on the satellite connection).
- Separate or built-in speakers and microphone
- A computer with at least a 1GHz processor, 256 MB RAM and a webcam is needed to make video calls
- At least a 128 MB RAM, a 400 MHz processor, and 50 MB free disk space on the hard drive

Steps for Opening a New Skype ID

- Click to say that you do not already have a Skype Name.
- Create a memorable Skype Name that you wish to use as your User Name.
- Think of a password. This should contain at least 4 characters and be difficult for other people to guess.
- You are legally required to read and agree to the **License Agreement**, the **Terms of Service,** and the **Privacy Statement.** Click on each to view them.

- Your alternate e-mail ID, if provided here, will be kept a secret. However, if people already know it, they can find you in the directory.
- Click on **Sign In** to start using Skype.
- You will now be signed in and the Getting Started guide will open up. (You might need to have permission for this guide to open.)

Steps for Chatting on Skype

- Click on the name of the person you wish to chat with and select the **Send message** icon.
- Set a chat topic by opening **Chat Options** and clicking on **Set Topic**. (You may omit this step and commence immediately, if you so desire).
- The main window will display your messages as well as your contact's replies.
- To chat with more than one person, click **Add more people to this chat**.
- Type your message in the chat box and press enter on your keyboard.
- Select the name of the person you wish to chat with and click **Add**. A Skype chat can include up to 100 participants.
- When you add a contact, they will appear in the **Chat participants** list.
- Once all the contacts have been added, click **OK.** You will now automatically return to the chat window.
- All the chat participants will now become visible to each other; you can send messages (and files) to the group.
- You can also bookmark a chat if you wish to return to it later. To do so, click on **Book** at the top of the chat window and select **Bookmark Chat**. Your **Bookmarked Chats** will now have this chat listed on it.
- To exit a chat room without getting alerts of new messages, click **Leave**.

Check Your Progress

1. What is World Wide Web (WWW)?
2. Who invented World Wide Web (WWW)?
3. What are search engines?
4. What is the importance of a web browser?
5. What is conferencing? How it is done?

2.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs), such as https://example.com/, which

NOTES

may be interlinked by hypertext, and are accessible over the Internet. The resources of the Web are transferred via the HyperText Transfer Protocol (HTTP) and may be accessed by users by a software application called a 'Web Browser' and are published by a software application called a 'Web Server'.

2. English engineer and computer scientist Sir Timothy John Berners-Lee invented the World Wide Web (WWW) in 1989. He wrote the first web browser in 1990 while employed at CERN near Geneva, Switzerland. The browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and then to the general public in August 1991. The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet.
3. Search engines are the software that enable searching of the content available on Internet. A search engine is an information retrieval system which is used to access and retrieve information stored in WWW or a computer system attached to the Internet. Search engines also help minimize the time required to find the relevant information on the computer system. The computer system could be a standalone system or it could also be attached to the Internet.
4. A browser is a software, which your computer uses to view WWW documents and access the Internet. The browser program residing in your computer provides you with the facilities like text formatting, hypertext links, images, sounds, motion and other features. Internet Explorer and Netscape are considered the most widely used browsers. Browsers have sub programs called plug-ins to handle the documents found on the Web. It may also have other plug-ins stored elsewhere in the computer.
5. Conferencing brings together groups of people to share their experiences, information, and expertise. Traditional conferences have required that people share the same physical space and time. With the advent of technology, and telecommunications, conferences no longer require a shared physical space, but still require a coordinated time for participants to meet. The Internet, and other technologies, support conferencing across spatial boundaries by connecting people electronically. There is a variety of Internet conferencing tools, including video cameras, whiteboard software, groupware tools, and Web conferencing products.

2.7 SUMMARY

- The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs), such as https://example.com/, which may be interlinked by hypertext, and are accessible over the Internet.

NOTES

- The resources of the Web are transferred via the HyperText Transfer Protocol (HTTP) and may be accessed by users by a software application called a 'Web Browser' and are published by a software application called a 'Web Server'.
- English engineer and computer scientist Sir Timothy John Berners-Lee invented the World Wide Web (WWW) in 1989. He wrote the first web browser in 1990 while employed at CERN near Geneva, Switzerland. The browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and then to the general public in August 1991.
- The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet.
- The web service provider helps establish a home page on the net. It is also known as electronic description of the organizations and its products and services. In real life, it is similar to a brochure or a catalog.
- An attractive home page grabs the attention of the visitors and that is why its cost per page is higher as compared to regular web pages. A home page can be setup with the help of Integrated Services Digital Network (ISDN) line as it generates HTML which represents the graphical interface.
- Search engines are the software that enable searching of the content available on Internet. A search engine is an information retrieval system which is used to access and retrieve information stored in WWW or a computer system attached to the Internet.
- Search engines also help minimize the time required to find the relevant information on the computer system. The computer system could be a standalone system or it could also be attached to the Internet.
- Search engines are the interface to a group of contents, which allow the users to type in the keywords, so that the engine can find several matching contents to the corresponding keywords out of millions of web pages. The keywords provided by the user are referred to as a search query.
- A browser is a software, which your computer uses to view WWW documents and access the Internet. The browser program residing in your computer provides you with the facilities like text formatting, hypertext links, images, sounds, motion and other features. Internet Explorer and Netscape are considered the most widely used browsers.
- Browsers have sub programs called plug-ins to handle the documents found on the Web. It may also have other plug-ins stored elsewhere in the computer.
- Web browsers are used to interpret special hypertext pages consisting of the HyperText Markup Language (HTML) and JavaScript so that they can be displayed in the given format. Some of the widely used web browsers are Internet Explorer, Netscape Navigator, Mozilla Firefox, Google Chrome, Lynx, Opera, Apple's Safari, etc.

NOTES

- Online chat may refer to any kind of communication over the Internet that offers a real-time transmission of text messages from sender to receiver.
- Chat messages are generally short in order to enable other participants to respond quickly. Thereby, a feeling similar to a spoken conversation is created, which distinguishes chatting from other text-based online communication forms, such as the Internet forums and e-mail.
- Online chat may address point-to-point communications as well as multicast communications from one sender to many receivers and voice and video chat, or may be a feature of a web conferencing service.
- Online chat may be primarily defined as any direct text-based or video-based (webcams), one-on-one chat or one-to-many group chat (formally also known as synchronous conferencing), using tools, such as instant messengers, Internet Relay Chat (IRC), talkers and possibly Multi-User Dungeons (MUDs).
- The expression online chat comes from the word chat which means 'Informal Conversation'. Online chat includes web-based applications that allow communication – often directly addressed, but anonymous between users in a multi-user environment. Web conferencing is a more specific online service, i.e., often sold as a service, hosted on a web server controlled by the vendor.
- Conferencing brings together groups of people to share their experiences, information, and expertise. Traditional conferences have required that people share the same physical space and time. With the advent of technology, and telecommunications, conferences no longer require a shared physical space, but still require a coordinated time for participants to meet.
- The Internet, and other technologies, support conferencing across spatial boundaries by connecting people electronically. There is a variety of Internet conferencing tools, including video cameras, whiteboard software, groupware tools, and Web conferencing products.
- Videoconferencing represents real-time moving images, usually at 30 frames per second (fps) or slower, broadcast and received over the Internet. Video is a powerful medium for human communication, since it supports visual and audio signals for social interaction. Video also requires a large bandwidth for transmission and reception, since each second of video represents between 10 and 30 frames.

2.8 KEY WORDS

- **World Wide Web (WWW):** The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs), such

as https://example.com/, which may be interlinked by hypertext, and are accessible over the Internet.

- **Search engines:** Search engines are the software that enable searching of the content available on Internet. A search engine is an information retrieval system which is used to access and retrieve information stored in WWW or a computer system attached to the Internet.
- **Client programs:** These are used to establish TCP sessions with the web server or the proxy server. To accomplish this task, the client programs make use of HTTP, FTP, Gopher or Telnet.
- **Browser:** A browser is a software, which your computer uses to view WWW documents and access the Internet, it residing in your computer and provides you with the facilities like text formatting, hypertext links, images, sounds, motion and other features. Internet Explorer and Netscape are considered the most widely used browsers.
- **Videoconferencing:** Video represents real-time moving images, usually at 30 frames per second (fps) or slower, broadcast and received over the Internet. Video is a powerful medium for human communication, since it supports visual and audio signals for social interaction. Video also requires a large bandwidth for transmission and reception, since each second of video represents between 10 and 30 frames.

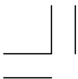
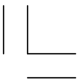
2.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the term World Wide Web (WWW).
2. What are Internet search engines?
3. Explain the importance of web browsers.
4. Explain how chatting and conferencing is done on the Internet.

Long-Answer Questions

1. Discuss the basic concepts of World Wide Web (WWW) with the help of appropriate examples.
2. Briefly explain the concept of Internet search engines giving appropriate examples.
3. Explain the structure and significance of Web Browsers for the Internet.
4. Why chatting and conferencing is becoming popular on the Internet? Discuss about the various software that support it.

2.10 FURTHER READINGS

NOTES

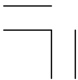
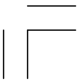
Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

UNIT 3 INTERNET CHATS, INTERNET RELAY CHAT (IRC), FTP AND TELNET

NOTES

Structure

3.0 INTRODUCTION

Internet is a network of computers linking many different types of computers all over the world. It is a network of networks sharing a common mechanism for addressing (identifying) computers, and a common set of communication protocols for communications between two computers on the network. The communications infrastructure of the Internet consists of its hardware components and a system of software layers that control various aspects of the architecture. Internet Service Providers (ISPs) establish the worldwide connectivity between individual networks at various levels of scope. End-users who only access the Internet when needed to perform a function or obtain information, represent the bottom of the routing hierarchy.

Internet chats not only allow you to send and receive instant messages, they also allow you to share pictures, and files. A 'Usenet Newsgroup' is a repository usually within the Usenet system, for messages posted from many users in different locations using Internet. They are discussion groups and are not devoted to publishing news.

Internet Relay Chat (IRC) is an application layer protocol that facilitates communication in the form of text. The chat process works on a client/server networking model. IRC clients are computer programs that users can install on their system or web based applications running either locally in the browser or on a 3rd party server. The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. The word 'Telnet' is derived from telecommunications and network. It

NOTES

is one of the oldest protocols in the TCP/IP suite, first developed in the 1960s. Telnet allows a user on one computer system to directly access and interact with remote computers anywhere on a network or the Internet. The connection between the local system and the remote system is referred to as the Telnet session.

In this unit, you will study about the online chatting and messaging, Usenet Newsgroup, Internet Relay Chat (IRC), FTP and Telnet.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the concept of online chatting and messaging
- Define the importance of the Usenet Newsgroup
- Discuss the significance of the Internet Relay Chat (IRC)
- Elaborate on the concept FTP and Telnet

3.2 ONLINE CHATTING AND MESSAGING

Internet chats not only allow you to send and receive instant messages, they also allow you to share pictures, and files. Some Internet chat rooms include emoticons which are smiley faces used to describe what your present emotion is. Some Internet chats include sound effects which range from serious to silly and allow you to. Other chat rooms allow you to change colour combinations to create a theme or background that works for you as you are chatting.

Chatting online refers to talking to someone or many persons using the Internet and sending the typed messages back and forth. You type your message and send it, other people read it and type their messages and send them. You read the messages, type your reply and send it back, and so on. Each person's messages are identified by the screen name of the person who typed it. A screen name is the name with which you are known by to this chat group. You may use the same screen name each time, or choose a different one for each chat session. For security reasons, do not use your real name. The people you are chatting to may use different names as well.

You can chat in two ways either using a channel or using a direct connection.

Channel: A channel is where a lot of people are talking together. It is another name for a chat room. Each channel has a name that should indicate what the people in the room are talking about.

Direct Connection: A direct connection is a private conversation between you and another person using the Internet. You join chat groups in different ways, depending on what sort of ISP you have. If you have a PPP account with your ISP, you can use the IRC (Internet Relay Chat) to chat or talk.

When you first join a chat group, you will see the screen names of people who are already chatting and a window that is keeping track of their messages. Everyone in the group will be notified when you join it, so if it is friendly someone should send you a welcome message.

NOTES

Writing Your Messages

When you are chatting, you use smileys to express your feelings. Common smileys are listed in the section on E-Mail. People also use simple abbreviations. Some are easy to understand, for example u for you). Others are acronyms.

Things to Remember When You Join a Chat Group

1. The conversation has already been going for a while, therefore read a full page of exchanges (messages) before you type any message so that you understand what people are talking about.
2. Start by reading the messages from one person, then another person, etc., until you get used to following all the different threads of the conversation.
3. Scroll up to read older messages if you have to, and scroll down to see new messages.
4. Keep your messages short and to the point.
5. If you like, use the chat system to create a profile about yourself that other chatters can read (as you can read theirs). Do not include any personal information.
6. If you have someone in your chat group being offensive either ignore them, go to another chat room or set your screen so you do not get their messages.

Things to Remember to Chat Safely

1. Many people lie about the information. This includes personal information also.
2. Do not reveal information that enables someone to find you, such as name, address, phone number, place of work, suburb, mailing address, etc.
3. Never give your password to anyone.
4. Be very cautious if someone without a profile wants to chat with you.

3.3 USENET NEWSGROUP

On the Internet, you can also share news, views, ideas and information on any topic like politics, science, computers, etc. This can be accomplished using a newsgroup service. A Newsgroup, also known as Usenet, is an online discussion group on the Internet who share a common interest. Many websites like *www.NewsOne.net, www.google.com, www.Usenet-Replayer.com,* etc. provide the facility of newsgroups.

People interested in a specific topic, write articles and post them to the bulletin board of the newsgroup server (a machine that posts the message for everyone to read) so that others can read, reply and comment on them. You can read and post newsgroup articles using newsreader software such as Microsoft Outlook Express News or Netscape News or using a web browser like Internet Explorer.

A 'Newsgroup' is a discussion about a particular subject consisting of notes written to a central Internet site and redistributed through 'Usenet', a worldwide network of news discussion groups. Usenet uses the Network News Transfer Protocol (NNTP).

A 'Usenet Newsgroup' is a repository usually within the Usenet system, for messages posted from many users in different locations using Internet. They are discussion groups and are not devoted to publishing news. Newsgroups are technically distinct from, but functionally similar to, discussion forums on the World Wide Web (WWW). Newsreader software is used to read the content of newsgroups.

Before the adoption of the World Wide Web, Usenet newsgroups were among the most popular Internet services, and have retained their non-commercial nature. In recent years, this form of open discussion on the Internet has lost considerable ground to individually-operated browser-accessible forums and big media social networks, such as Facebook and Twitter.

Communication is facilitated by the Network News Transfer Protocol (NNTP) which allows connection to Usenet servers and data transfer over the Internet. Similar to another early (yet still used) protocol SMTP which is used for email messages. NNTP allows both server-server and client-server communication - this means that newsgroups can be replicated from server to server which gives the Usenet network the ability to maintain a level of robust data persistence as a result of built-in data redundancy. However, most users will access using only the client-server commands of NNTP and in almost all cases will use a GUI for browsing as opposed to command line based client-server communication specified in the NNT protocol.

Newsgroups generally come in either of two types, binary or text. There is no technical difference between the two, but the naming differentiation allows users and servers with limited facilities to minimize network bandwidth usage. Generally, Usenet conventions and rules are enacted with the primary intention of minimizing the overall amount of network traffic and resource usage. Typically, the newsgroup is focused on a particular topic of interest. A message sent for publication on a newsgroup is called a 'post'. Some newsgroups allow posts on a wide variety of themes, regarding anything a member chooses to discuss as on-topic, while others keep more strictly to their particular subject, frowning on off-topic posts. The news admin (the administrator of a news server) decides how long posts are kept on their server before being expired (deleted). Different servers will have different retention times for the same newsgroup; some may keep posts for as little as one or two weeks, others may hold them for many months. Some admins keep posts in local or technical newsgroups around longer than posts in other newsgroups.

3.4 INTERNET RELAY CHAT (IRC)

Internet Relay Chat (IRC) is an application layer protocol that facilitates communication in the form of text. The chat process works on a client/server networking model. IRC clients are computer programs that users can install on their system or web based applications running either locally in the browser or on a 3rd party server. These clients communicate with chat servers to transfer messages to other clients. IRC is mainly designed for group communication in discussion forums, called channels, but also allows one-on-one communication via private messages as well as chat and data transfer, including file sharing.

Client software is available for every major operating system that supports Internet access. As of April 2011, the top 100 IRC networks served more than half a million users at a time, with hundreds of thousands of channels operating on a total of roughly 1,500 servers out of roughly 3,200 servers worldwide.

IRC was created by Jarkko Oikarinen in August 1988 to replace a program called MUT (MultiUser Talk) on a BBS called OuluBox at the University of Oulu in Finland, where he was working at the Department of Information Processing Science. Jarkko intended to extend the BBS software he administered, to allow NEWS in the Usenet style, real time discussions and similar BBS features. Oikarinen found inspiration in a chat system known as Bitnet Relay, which operated on the BITNET. In November 1988, IRC had spread across the Internet and in the middle of 1989, there were some 40 servers worldwide.

IRC is an open protocol that uses TCP and, optionally, TLS. An IRC server can connect to other IRC servers to expand the IRC network. Users access IRC networks by connecting a client to a server. There are many client implementations, such as mIRC, HexChat and irssi, and server implementations, for example the original IRCd. Most IRC servers do not require users to register an account but a nick is required before being connected.

IRC was originally a plain text protocol (although later extended), which on request was assigned port 194/TCP by IANA. However, the de facto standard has always been to run IRC on 6667/TCP and nearby port numbers (for example TCP ports 6660–6669, 7000) to avoid having to run the IRCd software with root privileges.

The protocol specified that characters were 8-bit but did not specify the character encoding the text was supposed to use. This can cause problems when users using different clients and/or different platforms want to converse.

All client-to-server IRC protocols in use today are descended from the protocol implemented in the irc2.4.0 version of the IRC2 server, and documented in RFC 1459. Since RFC 1459 was published, the new features in the irc2.10 implementation led to the publication of several revised protocol documents (RFC 2810, RFC 2811, RFC 2812 and RFC 2813); however, these protocol changes have not been widely adopted among other implementations.

NOTES

Microsoft made an extension for IRC in 1998 via the proprietary IRCX. They later stopped distributing software supporting IRCX, instead developing the proprietary MSNP.

The standard structure of a network of IRC servers is a tree. Messages are routed along only necessary branches of the tree but network state is sent to every server and there is generally a high degree of implicit trust between servers. However, this architecture has a number of problems. A misbehaving or malicious server can cause major damage to the network and any changes in structure, whether intentional or a result of conditions on the underlying network, require a Net-Split and Net-Join. This results in a lot of network traffic and spurious quit/join messages to users and temporary loss of communication to users on the splitting servers. Adding a server to a large network means a large background bandwidth load on the network and a large memory load on the server. Once established, however, each message to multiple recipients is delivered in a fashion similar to multicast, meaning each message travels a network link exactly once. This is a strength in comparison to non-multicasting protocols such as Simple Mail Transfer Protocol (SMTP) or Extensible Messaging and Presence Protocol (XMPP).

An IRC daemon can also be used on a Local Area Network (LAN). IRC can thus be used to facilitate communication between people within the local area network (internal communication).

3.5 FTP

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network.

FTP is built on a client-server model architecture using separate control and data connections between the client and the server. FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS) or replaced with SSH File Transfer Protocol (SFTP).

The first FTP client applications were command-line programs developed before operating systems had graphical user interfaces, and are still shipped with most Windows, UNIX, and Linux operating systems. Many FTP clients and automation utilities have since been developed for desktops, servers, mobile devices, and hardware, and FTP has been incorporated into productivity applications, such as HTML editors.

Communication and Data Transfer via FTP

FTP may run in active or passive mode, which determines how the data connection is established. This sense of 'mode' is different from that of the MODE command in the FTP protocol, and actually corresponds to the PORT/PASV/EPSV/, etc.,

NOTES

commands instead. In both cases, the client creates a TCP control connection from a random, usually an unprivileged, Port N to the FTP server command Port 21.

- In active mode, the client starts listening for incoming data connections from the server on Port M. It sends the FTP command Port M to inform the server on which port it is listening. The server then initiates a data channel to the client from its Port 20, the FTP server data port.
- In situations where the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used. In this mode, the client uses the control connection to send a PASV command to the server and then receives a server IP address and server port number from the server, which the client then uses to open a data connection from an arbitrary client port to the server IP address and server port number received.

Both modes were updated in September 1998 to support IPv6.

The server responds over the control connection with three-digit status codes in ASCII with an optional text message. For example, '200' or '200 OK' means that the last command was successful. The numbers represent the code for the response and the optional text represents a human-readable explanation or request, for example <Need account for storing file>). An ongoing transfer of file data over the data connection can be aborted using an interrupt message sent over the control connection.

The reason why FTP needs two ports (one for sending and one for receiving) is due to the fact that it was originally designed to operate on Network Control Program (NCP), which was a simplex protocol that utilized two port addresses, establishing two connections, for two-way communications. An odd and an even port were reserved for each application layer application or protocol. The standardization of TCP and UDP reduced the need for the use of two simplex ports for each application down to one duplex port, but the FTP protocol was never altered to only use one port, but continued using two for backwards compatibility.

FTP normally transfers data by having the server connect back to the client, after the PORT command is sent by the client. This is problematic for both NATs and firewalls, which do not allow connections from the Internet towards internal hosts. For NATs, an additional complication is that the representation of the IP addresses and port number in the PORT command refer to the internal host's IP address and port, rather than the public IP address and port of the NAT.

3.6 TELNET

The word 'Telnet' is derived from telecommunications and network. It is one of the oldest protocols in the TCP/IP suite, first developed in the 1960s. Telnet allows a user on one computer system to directly access and interact with remote computers anywhere on a network or the Internet. The connection between the local system and the remote system is referred to as the Telnet session.

NOTES

All the users on a network or on the Internet are not allowed to access any remote machine. To have access to a remote computer, you must have the permission to use it. The computer that wants to access the remote computer is called the Telnet client and the remote computer that acts as the host is called the Telnet server. The TCP/IP protocol is used to transmit information between the Telnet client and the Telnet server. Telnet is text-based and only the keyboard can be used for navigation. For this reason, it is popularly used in UNIX-based systems.

Telnet creates a standard and a fictional terminal called the Network Virtual Terminal (NVT) that is used for communication by all the computers on the network. The steps that are performed while conducting a Telnet session are as follows:

- The inputs from the user are taken and translated by the Telnet client to the NVT form.
- This input is sent to a Telnet server running on a remote computer.
- The server translates the inputs from NVT to whatever representation the computer being accessed requires.

The same steps are repeated when data is sent from the remote computer back to the user. This system allows clients and servers to communicate even if they use entirely different hardware and internal data representations.

Check Your Progress

1. What does chatting online?
2. Define the term Usenet Newsgroup.
3. What is Internet Relay Chat (IRC)?
4. What is File Transfer Protocol (FTP)?
5. Explain the term Telnet.

3.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Chatting online refers to talking to someone or many persons using the Internet and sending the typed messages back and forth. You type your message and send it, other people read it and type their messages and send them. You read the messages, type your reply and send it back, and so on. Each person's messages are identified by the screen name of the person who typed it.
2. A 'Usenet Newsgroup' is a repository usually within the Usenet system, for messages posted from many users in different locations using Internet. They are discussion groups and are not devoted to publishing news. Newsgroups are technically distinct from, but functionally similar to, discussion forums on the World Wide Web (WWW). Newsreader software is used to read the content of newsgroups.

3. Internet Relay Chat (IRC) is an application layer protocol that facilitates communication in the form of text. The chat process works on a client/server networking model. IRC clients are computer programs that users can install on their system or web based applications running either locally in the browser or on a 3rd party server.
4. The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. FTP is built on a client-server model architecture using separate control and data connections between the client and the server.
5. The word 'Telnet' is derived from telecommunications and network. It is one of the oldest protocols in the TCP/IP suite, first developed in the 1960s. Telnet allows a user on one computer system to directly access and interact with remote computers anywhere on a network or the Internet. The connection between the local system and the remote system is referred to as the Telnet session.

3.8 SUMMARY

- Internet chats not only allow you to send and receive instant messages, they also allow you to share pictures, and files.
- Chatting online refers to talking to someone or many persons using the Internet and sending the typed messages back and forth. You type your message and send it, other people read it and type their messages and send them. You read the messages, type your reply and send it back, and so on.
- Each person's messages are identified by the screen name of the person who typed it.
- A screen name is the name with which you are known by to this chat group. You may use the same screen name each time, or choose a different one for each chat session. For security reasons, do not use your real name. The people you are chatting to may use different names as well.
- You can chat in two ways either using a channel or using a direct connection.
- A channel is where a lot of people are talking together. It is another name for a chat room. Each channel has a name that should indicate what the people in the room are talking about.
- A direct connection is a private conversation between you and another person using the Internet. You join chat groups in different ways, depending on what sort of ISP you have. If you have a PPP account with your ISP, you can use the IRC (Internet Relay Chat) to chat or talk.
- On the Internet, you can also share news, views, ideas and information on any topic like politics, science, computers, etc. This can be accomplished using a newsgroup service.

NOTES

- A 'Newsgroup' is a discussion about a particular subject consisting of notes written to a central Internet site and redistributed through 'Usenet', a worldwide network of news discussion groups. Usenet uses the Network News Transfer Protocol (NNTP).
- A 'Usenet Newsgroup' is a repository usually within the Usenet system, for messages posted from many users in different locations using Internet. They are discussion groups and are not devoted to publishing news.
- Newsgroups are technically distinct from, but functionally similar to, discussion forums on the World Wide Web (WWW). Newsreader software is used to read the content of newsgroups.
- Communication is facilitated by the Network News Transfer Protocol (NNTP) which allows connection to Usenet servers and data transfer over the Internet.
- NNTP allows both server-server and client-server communication - this means that newsgroups can be replicated from server to server which gives the Usenet network the ability to maintain a level of robust data persistence as a result of built-in data redundancy.
- Newsgroups generally come in either of two types, binary or text. There is no technical difference between the two, but the naming differentiation allows users and servers with limited facilities to minimize network bandwidth usage.
- Internet Relay Chat (IRC) is an application layer protocol that facilitates communication in the form of text. The chat process works on a client/server networking model.
- IRC clients are computer programs that users can install on their system or web based applications running either locally in the browser or on a 3rd party server. These clients communicate with chat servers to transfer messages to other clients.
- IRC is an open protocol that uses TCP and, optionally, TLS. An IRC server can connect to other IRC servers to expand the IRC network. Users access IRC networks by connecting a client to a server.
- The standard structure of a network of IRC servers is a tree. Messages are routed along only necessary branches of the tree but network state is sent to every server and there is generally a high degree of implicit trust between servers.
- The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network.
- FTP is built on a client-server model architecture using separate control and data connections between the client and the server.

- FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.
- For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS) or replaced with SSH File Transfer Protocol (SFTP).
- The word 'Telnet' is derived from telecommunications and network. It is one of the oldest protocols in the TCP/IP suite, first developed in the 1960s.
- Telnet allows a user on one computer system to directly access and interact with remote computers anywhere on a network or the Internet. The connection between the local system and the remote system is referred to as the Telnet session.
- Telnet creates a standard and a fictional terminal called the Network Virtual Terminal (NVT) that is used for communication by all the computers on the network.

3.9 KEY WORDS

- **Channel:** A channel is where a lot of people are talking together. It is another name for a chat room. Each channel has a name that should indicate what the people in the room are talking about.
- **Direct connection:** A direct connection is a private conversation between you and another person using the Internet.
- **Usenet Newsgroup:** A 'Usenet Newsgroup' is a repository usually within the Usenet system, for messages posted from many users in different locations using Internet, they are discussion groups and are not devoted to publishing news.
- **Internet Relay Chat (IRC):** Internet Relay Chat (IRC) is an application layer protocol that facilitates communication in the form of text, the chat process works on a client/server networking model.
- **File Transfer Protocol (FTP):** The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. FTP is built on a client-server model architecture using separate control and data connections between the client and the server.
- **Telnet:** The word 'Telnet' is derived from telecommunications and network, it is one of the oldest protocols in the TCP/IP suite, and allows a user on one computer system to directly access and interact with remote computers anywhere on a network or the Internet.

3.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. What is online chatting?
2. How is messaging done in online chatting?
3. Define the term Usenet Newsgroup.
4. Explain about Internet Relay Chat (IRC)
5. What is FTP?
6. Define the term Telnet.

Long-Answer Questions

1. Briefly discuss the concept of online chatting and messaging. What precautions must be taken during online chatting?
2. Discuss about the Usenet Newsgroup giving its feature and significance.
3. Explain the importance of Internet Relay Chat (IRC). What is the standard structure of a network of IRC servers?
4. Explain the communication and data transfer process via FTP.
5. Write the steps that are performed while conducting a Telnet session.
6. Write short notes on the following:
   (a) FTP
   (b) Telnet

3.11 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

BLOCK - II
FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING

NOTES

UNIT 4 BASIC CONCEPTS OF OOP

Structure

4.0 INTRODUCTION

Object Oriented Programming (OOP) is a paradigm that provides many concepts, such as inheritance, data binding, polymorphism, etc. There are four major principles that make any language 'Object Oriented'. The object oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Data Abstraction, Encapsulation etc. These are also termed as four pillars of OOP. Encapsulation is the mechanism of hiding of data implementation by restricting access to public methods. Instance variables are kept private and access or methods are made public to achieve this. Abstract means a concept or a notion which is not associated with any particular instance. Using abstract Class/Interface we express the intent of the class rather than the actual implementation. Inheritance concept is used to express a relationship between two objects. Using inheritance, in derived classes we can reuse the code of existing super classes. In Java, concept of inheritance is referred as 'is-a' based on class inheritance (using extends) or interface implementation (using implements). Polymorphism means one name for many forms. It is further of two types — static and dynamic.

In this unit, you will study about the basic concepts of OOP, benefits, applications, Java evolution and features, how Java differs from C and C++, Java and the Internet, Java support system and Java environment.

NOTES

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the history and features of Java
- Describe the significance of bytecode
- Compare Java and C++
- Explain the various keywords of Java
- Distinguish between valid and invalid Java identifiers
- Understand the basic concepts of object oriented programming
- Learn about the benefits of object oriented programming

4.2 OVERVIEW OF JAVA

If you already know C, you only have to make a little more effort to become a Java expert. However, if you have learned C++, you have to unlearn a little before you proceed to learn Java.

Java, which is an object oriented programming language is based on the concept of an object. If you have some familiarity with an object oriented language, such as Smalltalk or C++, much of Java will be familiar to you. Java is derived from C++, but is slightly simplified with libraries and convenient for the Internet. It is the programming language for the Internet environment. The Internet implies heterogeneous systems, different network features, different windows libraries and different operating systems. Java guarantees identical program behaviour on different platforms.

Java is an interpreted language. Java compiler complies the Java source code to obtain an object code termed as bytecode in Java terminology. The Java interpreter will interpret and run by the byte code at runtime.

Java is architectural neutral language. Architectural neutrality means that the bytecode that is the output of the Java compiler will run on any processor and operating system. You can compile a Java program to obtain the bytecode. This bytecode or object code will run on any processor and operating system, provided a suitable interpreter is available.

Java is portable as well. This means that Java provides the same Application Programmer's Interface (API) functions or library calls on each system. Java API functions are the same for a programmer on Window 95, Windows NT or on

Solaris 2.x. To port a program from one of these platforms to another, what the programmer generally does is to recompile the program. However, due to the architectural neutrality of Java, you do not even have to do that.

Why Java is Important to the Internet?

Java came to the forefront of programming with the help of the Internet. Consequently, Java had a deep effect on the Internet. In addition to simplifying Web programming in general, Java innovated a new type of networked program called the applet. A Java applet is a small program embedded in a Web page which runs when that page is browsed using a Web browser.

Java's Magic: The Bytecode

You have just learned that the output of a Java compiler is not an executable code. Rather, it is a bytecode. In Java bytecode is a highly optimized set of instructions planned to be executed by the Java runtime system, which is referred to as the Java Virtual Machine (JVM). A Java program executed by the JVM helps solve problems associated with Web based programs. Since the JVM is in control, it can hold the program and check it from generating side effects outside the system.

Java Buzzwords

As mentioned earlier, portability and security is the innovation of Java. Some additional factors too played significant roles in moulding the final form of Java. The list of buzzwords summed up by the Java team is as follows:

- **Simplicity:** Programmers find it easy to write applications as it avoids the use of the concepts of C/C++, such as pointers, operator overloading, multiple inheritance, etc. String manipulations can be implemented very easily without any explicit concatenation procedure.
- **Portability:** Java is famous for its unique feature—platform independence (architecture neutral). This means that a Java program compiled on one machine could be ported to any other machine/operating system and executed without any modifications. Thus, the class file, which is the result of compilation resides on a DOS platform, can run on a UNIX platform unlike the .exe file of C/C++. This is an important feature for which Java is preferred for the Internet applications.
- **Robust and Object Oriented:** Java is a highly object oriented language where reusability is of utmost importance. Java programs are very reliable on different platforms with the special features of memory allocation and de-allocation, and exception handling. Memory management, especially de-allocation, is taken care of by the Java environment, which is not the case in C/C++, where programmers have to deal with it explicitly with extra code. Exception handling is a mechanism that helps the execution of a code, even if an error occurs at runtime, by handling the exception.

NOTES

Robustness is also achieved because Java is a strongly typed language. It signifies that you must declare the variable type before you use it. This is different from languages, such as PERL, JavaScript, etc., which are loosely typed.

- **Multithreaded:** Java with its multithreaded approach can run many programs concurrently, thereby saving processor time. Synchronization of code is an added feature of Java to run non-erroneous interactive applications.
- **Distributed and Dynamic:** Java is also popular for its distributed environment, as it supports the Transmission Control Protocol/Internet Protocol (TCP/IP). With Java, you can access a Uniform Resource Locator (URL) or a file on a remote server in some other country with the same ease, as you can access a file on your local system.

Java can also validate the code at runtime, which is more important for applets. Therefore, it is feasible to dynamically connect the code in a secure and practical manner.

4.3 JAVA EVOLUTION AND FEATURES

In 1991, a group of engineers at Sun Microsystems Laboratory developed the Java language. The Sun Microsystems engineers, Patrick Naughton and James Gosling wanted to design a programming language for the development of software that could be used in different consumer electronic devices, such as VCRs, toasters and cable television switch boxes. The team of Sun Microsystems engineers designed a project, which they named as Green Project.

The Green Project engineers designed a portable language that was small in nature, as its memory requirement was less and it can also be used to generate an intermediate code for virtual machines, such as Java Virtual Machine. The intermediate code generated by the Green Project required an interpreter to convert the programming language instructions into a machine code. This process of generating and interpreting the intermediate code required less memory space and thereby, provided compatibility to a variety of consumer electronic devices. The newly developed language was named Oak and was based on C++ and followed the object oriented programming approach. The Oak programming language was later renamed as Java and was introduced with this new name in 1995. In 1993, the first graphical Web browser Mosaic was released in the market. During this time the users across the world starting exploring the Internet. The users were using the Internet on their machines, which run on variety of OS (Operating Systems) and hardware configuration.

Due to the increasing popularity of the Web, the Sun engineers decided to use Java for the Internet by utilizing its feature of platform independence.

Patrick Naughton and Jonathan Payne built the HotJava Web browser written in the Java language. This Web browser was made capable of interpreting the intermediate code for Java.

NOTES

Features of Java

Java inherits most of its features from the earlier object oriented languages like C++. Various features of Java are:

- Compiled and interpreted.
- Platform independent and portable.
- Object oriented programming approach.
- Robust and secure.
- Simple and distributed.
- Small and familiar.
- Multithreaded and interactive.
- High performance.
- Dynamic and extensible.

Compiled and Interpreted

Java is a compiled and interpreted language. Java compiler translates the program source code into a compiled format known as bytecode instructions. The bytecode instructions can be executed by using Java Virtual Machine (JVM), on any system irrespective of the hardware configuration and operating system of the system in use. JVM is an abstract computing machine that is used for interpreting compiled bytecode instructions. Java interpreter converts the bytecode instructions into the machine code that can easily be understood by the machine.

Platform Independent and Portable

A Java program has a unique feature of portability. The Java programs can be easily transferred from one computer to another and can be executed anywhere and anytime. Java provides portability as the Java compiler generates the bytecode instructions that you can implement on any machine irrespective of the operating system or the processor.

Object Oriented Approach

Java is an object oriented language that supports various Object Oriented Programming (OOP) features, such as polymorphism, data abstraction and inheritance. Java enables you to store the program code and data in the form of objects and classes. Java contains various sets of classes that are organized in packages which you can use while executing a Java program.

NOTES

Robust and Secure

Java also supports the feature of exception handling which detects the errors and exceptions and reduces the possibility of runtime errors. Java ensures that the downloaded applet is virus free. Applet is a small Java program that executes in a Web browser, but can be executed in other applications or devices that support the Applet programming model. Java also supports the feature of garbage collection, which solves the memory management problem by deleting unused objects from the memory. Java is a secured language as it performs various checks on the programming code during runtime and compile time and ensures that the code should be reliable and bug free.

Simple and Distributed

The syntax for the Java language is similar to that of the C++ and C. The end user, who is familiar with C and C++, requires less effort in learning Java because, Java and C or C++ have similar syntax. Java is a simpler language than the existing languages and does not implement some of the redundant and complex concepts of C and C++. For example, Java does not support the concept of pointers, as pointers increase the complexity of programming. Java has a simple programming approach, as Java does not support the concept of multiple inheritance and operator overloading. However, Java supports multiple inheritance through interfaces which provides a structure of methods and variables to be used for defining classes. Java allows you to remotely access the Java applications from the Web. Java has the capability to handle various protocols, such as Transmission Control Protocol (TCP) and Internet Protocol (IP). These protocols enable you to access the information and application in the distributed environment of the Internet. You can also create distributed applications in Java that share resources, such as data and program from other systems connected to another network.

Small and Familiar

OOPs concept of inheritance allows the programmers and developers to reuse the predefined classes and objects in the programming code. Reusability of code in the programs makes programming simple and easier. The use of same programming code for other functions of an application develops familiarity in programs.

Multithreaded and Interactive

Java supports the process of multithreading that allows you to simultaneously perform multiple tasks. Multithreading allows you to start multiple tasks that executes on different threads. To perform multiple tasks, the Java runtime environment supports multi-process synchronization.

Multi tasking in Java allows the programmers to create applications that can simultaneously use audio-visual effects. Use of multimedia in Java makes the Java applications even more interesting.

High Performance

Java enables the developer to create high performance applications as the programs are first converted into bytecode and that bytecode is easily translated into machine language. Java programs require less memory and executes fast that also increase its performance.

Dynamic and Extensible

Java is more dynamic than C++ or C language because it allows dynamic linking of classes to libraries and predefined methods and objects. You can use programs written in other programming languages, such as C and C++ in Java, by dynamically linking them to the Java runtime environment. Java runtime environment makes inter-connections between the modules of a program at runtime. You can also add new methods and temporary variables in the existing libraries.

Java Environment

The Java environment comprises a set of tools and classes that are used to run the Java program. These tools are:

- Java development kit and
- Java standard library

Java Development Kit

Java Development Kit contains various tools used by the Java Runtime Environment (JRE) that can be used to compile and interpret Java programs. The Java tools include Java compiler, Java interpreter, Java disassembler and Java debugger. JRE is a software that is used to execute the Java program. The Java Development Kit (JDK) contains following tools to run and execute the Java programs:

- `appletviewer` allows you to run Java applets.
- `javac` is a Java compiler that translates the Java source code into Javaclass files or the bytecode files that can be interpreted by the java interpreter.
- `java` is a Java interpreter that executes various applications by interpreting the bytecodes files.
- `javah` is used for including the C header files in a Java program.
- `javap` acts as a disassembler and is used to convert bytecode files into a program source code.
- `javadoc` creates documentation of the Java source code files in the HTML format from the comments available in the java source code.
- `jdb` is a java debugger that locates errors in a Java program.

Figure 4.1 shows the basic configuration of the Java tools that is used to process a Java program.

NOTES

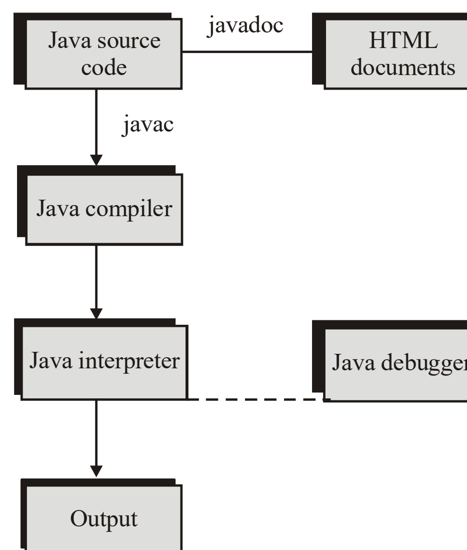


Fig. 4.1 *Implementation of the JDK Tools*

Java Standard Library

Java standard library is one of the most attractive features of Java that contain various classes to support all the major functions of Java. Java standard library includes the following classes:

- Language support classes which support basic features in Java, such as strings, arrays, threads and exception handling.
- Utility classes that are used for various functions, such as random number generator function, date and time function and container classes function.
- Input or output classes that are used to read data from various input devices like keyboard and joysticks and write data on various output devices, such as printer and monitor.
- Networking classes which allow communication between different computers over a network.
- Abstract Window Toolkit (AWT) that is used for creating platform independent GUI applications.
- Applet that allows you to create applets which can be downloaded and run on a client browser.

Java Virtual Machine

Java uses both compiler and interpreter. The source code written in Java is compiled to generate bytecode and then this bytecode is interpreted to machine instructions

NOTES

for a specific machine. The bytecode generated by the compiler is not machine specific as shown in Figure 4.2. It is generated for a virtual machine known as JVM (Java Virtual Machine). It exists only inside the computer memory. This virtual machine is designed in such a way that it can be implemented on any existing processor and itself acts as a virtual processor chip. It hides the underlying operating system details from Java applications.

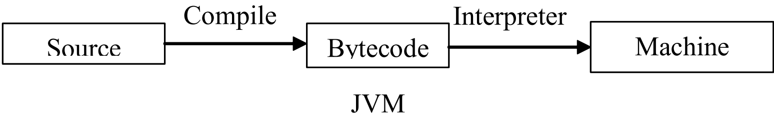


Fig. 4.2 *Execution of a Java Program*

4.4 OBJECT ORIENTED CONCEPTS

To understand the concept of object oriented programming, it is necessary to know the fundamental terms and concepts of this approach. These include objects, classes, data abstraction, encapsulation, inheritance, polymorphism and message passing.

Objects

Objects are small, self contained and modular units with a well defined boundary. An object consists of a state and behaviour. The state of an object is one of the possible conditions that an object can exist in and is represented by its characteristics or attributes or data. The behaviour of an object determines how an object acts or behaves and is represented by the operations that it can perform. In OOP, the attributes of an object are represented by the variables and the operations are represented by the functions.

For example, an object Biscuit may consist of data product code P001, product name Britania Biscuits, price 20 and quantity in hand 50. These data values specify the attributes or features of the object. Similarly, consider another object Maggi with product code P002, product name Maggi Noodles, price 10 and quantity in hand 20. In addition, the data in the object can be used by the functions, such as `check_qty()` and `display_product()`. These functions specify the actions that can be performed on data. Figure 4.3 shows how class and its objects are represented.

NOTES

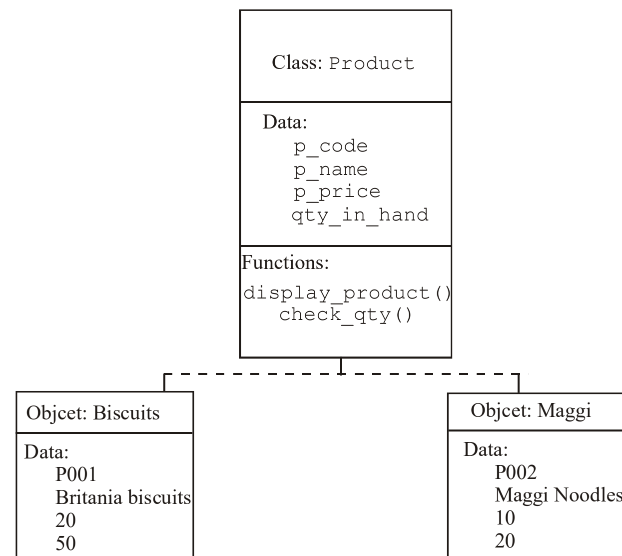


Fig. 4.3 *Class and its Objects*

Objects are the defined modules that are the basic runtime entities in object oriented systems. They are the building blocks of object oriented programming. Although, two or more objects can have same attributes, still they are separate and independent objects with their own identity. In other words, all the objects in a system take a separate space in the memory, independent of each other. The main objective of breaking down complex software projects into objects is that changes made to one part of a software should not adversely affect the other parts.

Classes

A class is defined as a user defined data type which contains the entire set of similar data and the functions that the objects possess. In other words, a class in OOP represents a group of similar objects. As stated earlier, in the real world millions of objects exist and each of them has its own identity. However, each of them can be categorized under different groups depending on the common properties they possess and the functions they perform. For example, cars, scooters, motorbikes, buses, etc. all can be grouped under the category 'vehicles'. Similarly, dogs, cats, horses etc. can be grouped under the category 'animals'. Thus, vehicles and animals can be considered as the classes.

A class serves as a blueprint or template for its objects. That is, once a class has been defined, any number of objects belonging to that class can be created. The objects of a class are also known as the instances or variables of that class and the process of creating objects from a class is known as instantiation. A class does not represent an object, rather it represents the data and functions that an object will have.

For example, a class Product consists of data, such as p_code, p_name, p_price and qty_in_hand which specify the attributes or features of the objects of the Product class. In addition, it consists of functions, such as display_product() and check_qty() that specify the actions that can be performed on data.

The data belonging to a particular class is known as its data members and the functions of the class are known as the member functions and both collectively are known as the members of the class.

Note: In Java, the data members are referred as instance variables and member functions are referred to as methods.

Abstraction

Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time. It allows the management of complex systems by concentrating on the essential features only. For example, while driving a car, a driver only knows the essential features to drive a car, such as how to use the clutch, brake, accelerator, gears, steering, etc. and is least bothered about the internal details of the car like the motor, engine, wiring etc.

Abstraction can be of two types— data abstraction and control abstraction. Data abstraction (also known as data hiding) means hiding the details about the data and control abstraction means hiding the implementation details. In object oriented approach, one can abstract both data and functions. However, generally the classes in OOP are defined in such a way that the data is hidden from the outside world and the functions form the public interface. That is, the functions of the class can be directly accessed by other functions outside the class and the hidden data can be accessed indirectly with the help of these functions.

Since the internal details of the class are hidden from the outside world, the data abstraction ensures security of the data by preventing it from accidental changes or manipulations by other parts of the program.

Note: Classes in the object oriented programming are also known as Abstract Data Types (ADT) as they use the concept of abstraction.

Encapsulation

Encapsulation is the technique of binding or keeping the data and functions (that operate on them) together in a single unit called a class. Encapsulation is the way to implement data abstraction. A well encapsulated object acts as a 'black box' for other parts of the program. That is, it provides services to the external functions or other objects that interact with it. However, these external functions or the objects do not need to know its internal details. For example, in Figure 1.3 the data p_code, p_name, p_price and qty_in_hand and the functions display_product() and check_qty are encapsulated in a class Product.

NOTES

Inheritance

Inheritance can be defined as the process where an object of a class acquires characteristics from the object of another class. As stated earlier, all the objects of a similar kind are grouped together to form a class. However, sometimes a situation arises when different objects cannot be combined together under a single group as they share only few common characteristics. In this situation, the classes are defined in such a way that the common features are combined to form a generalized class and the specific features are combined to form a specialized class. The specialized class is defined in such a way that in addition to the individual characteristics and functions, it also inherits all the properties and the functions of its generalized class.

The main advantage of inheritance is reusability. The existing classes can be simply re-used in new software instead of writing a new code. Moreover, new features can be added without altering or modifying the features of the existing class.

Polymorphism

Polymorphism (a Greek word meaning 'having multiple forms') is the ability of an entity, such as a function or a message to be processed in more than one form. It can also be defined as the property of an object belonging to a same or different class to respond to the same message or function in a different way. For example, if a message `change_gear` is passed to all the vehicles then the automobiles will respond to the message appropriately but the pulled vehicles will not respond. The concept of polymorphism plays an important role in OOP as it allows an entity to be represented in various forms.

Message Passing

Message passing is the process of interacting between the different objects in a program. As discussed earlier, a program following the object oriented paradigm comprises of a set of objects each, with a set of data and functions. When the program is executed, these objects interact or communicate with each other by sending and receiving messages. The messages are exchanged by calling the member functions of the classes.

Any object of a class that wants to communicate with the object of another class requests the object to invoke the required member function of its class. This function call is different from the normal function call as in this case the sending object is sending a request for the execution of the function. However, the receiving object may or may not accept the request depending on whether the function forms the public interface or it is hidden from the outside world. Thus, this form of communication is called message sending and is not an ordinary function call.

NOTES

For example, consider two classes Product and Order. The object of the Product class can communicate with the object of the Order class by sending a request for placing order Figure 4.4 illustrates the process of message passing.

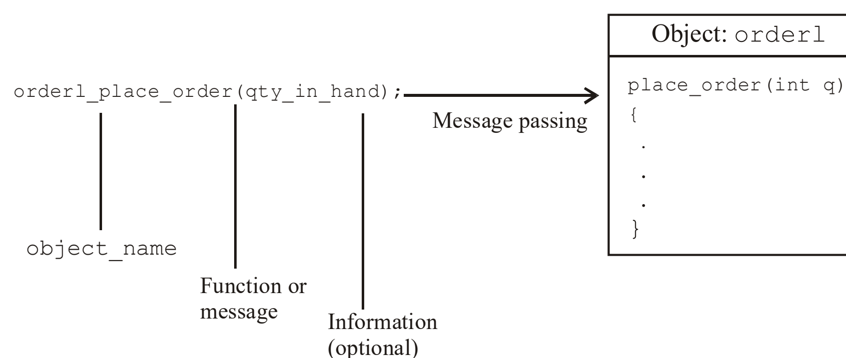


Fig. 4.4 *Message Passing*

Dynamic Binding

Dynamic binding is the process of linking a function call to the actual code of the function at runtime. That is, in dynamic binding, the actual code to be executed is not known to the compiler until runtime. The concept of dynamic binding is implemented with the help of inheritance and runtime polymorphism.

Consider an example in which three different classes, square, rectangle and circle are derived from the base class `geometrical_shapes`. The function `area()` of the base class is implemented in different ways in all its derived classes. At runtime, the desired function will be called depending on the object being referenced.

4.4.1 Benefits of OOP

The object oriented programming paradigm came into use owing to its ability to overcome certain limitations of structured and unstructured programming paradigms. The new and advanced features of OOP, such as encapsulation, abstraction, inheritance and polymorphism help in developing high quality software which has various advantages. Some of the advantages of OOP are as follows.

- In OOP, writing programs with the help of objects is similar to working with real world objects. That is, the real world objects can be conveniently represented in a program which reduces the compexity of the program and also makes the program structure clear.
- In object oriented programs, each object is an independent and separate entity which makes modifications, locates and fixes problems in a program an easy task. In addition, any changes made inside the class do not affect the other parts of a program. Thus, object oriented programs are easy to write and maintain.

NOTES

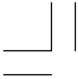
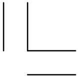
- In object oriented programming, data integrity and data security is high as it focuses on the data and its protection from manipulation by different parts of the program. As a result, object oriented programs are less prone to errors and more reliable and secure.
- Object oriented programs are easy to extend as new features in a program can be added easily by introducing some new objects without modifying the existing ones.
- Object oriented programming allows the reusability of codes. That is, the objects created in one program can be reused in other programs. In addition, new classes can be created with the help of existing ones using inheritance. It leads to faster software development and high quality programs.
- Object oriented programs are easier to adapt and scale, that is, large systems can be created by assembling reusable subsystems.

4.4.2 Applications of OOP

Since 1960, object oriented paradigm has touched many major application areas of software development. Some of the application areas where OOP has been used to develop software are as follows:

- **Simulation and Modelling:** Simulation is the technique of representing the real world entities with the help of a computer program. Simula-67 and Smalltalk are two object oriented languages are designed for making simulations.
- **User Interface Design:** Another popular application of OOP has been in the area of designing graphical user interfaces, such as Windows. C++ is mainly used for developing user interfaces.
- **Developing Computer Games:** OOP is also used for developing computer games, such as Diablo, Startcraft, Warcraft III and many more. These games offer virtual reality environments in which a number of objects interact with each other in complex ways to give the desired result.
- **Scripting:** In recent years, OOP has also been used for developing HTML, XHTML and XML documents for the Internet. Python, Ruby and Java are the scripting languages based on object oriented principles which are used for scripting.
- **Object Databases:** These days, OOP concepts have also been introduced in database systems to develop a new database management system named object database. These databases store the data directly in the form of objects. However, they are not as popular as the traditional relational database management systems.

Some other areas of applications include office automation systems, Decision Support Systems (DSS), Artificial Intelligence (AI) and expert systems, neural network, and parallel programming and Computer Aided Design (CAD) systems.

Check Your Progress

1. What is Java?
2. What are the types of Java programs?
3. What is the function of the Java interpreter?
4. Define abstraction.
5. What is inheritance?
6. What is primary advantage of reusability?
7. What is dynamic binding?

NOTES

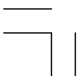
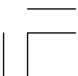
4.5 HOW JAVA DIFFERS FROM C++

Both C++ and Java are object-oriented languages but are very different form each other. Some of the features of C++ language were deliberately removed and some new features were added to make Java a more flexible and reliable language. Some of the features of Java that differentiate it from C++ are as follows:

- Java does not support the multiple inheritance of classes directly.
- The concept of multithreading is supported by Java.
- The destructor function in Java is replaced by the `finalize` method.
- The keyword `typdef` is not supported by Java.
- Java does not support pointers and instead it uses implicit object references.
- The virtual keyword is not supported in Java.
- Java does not support the concept of global variables.
- Java supports exception handling in a different way than C++. It provides the `finally` clause for cleanup.
- The non-primitive data types are allocated memory by using the new operator.
- Java adds many features that are necessary for object-oriented programming.

4.6 JAVA RUN-TIME ENVIRONMENT

As discussed earlier, Java uses both the compiler and interpreter. The source code written in Java is compiled to generate the bytecode and which is then interpreted to machine instructions for a specific machine. The bytecode generated by the compiler is not machine-specific. It is generated for the virtual machine that exists only inside the computer memory known as Java Virtual Machine (JVM). This virtual machine is designed in such a way that it can be implemented on the

top of any existing processor and itself acts as a virtual processor chip (Figure 4.5). It hides the underlying operating system details from Java applications.

NOTES

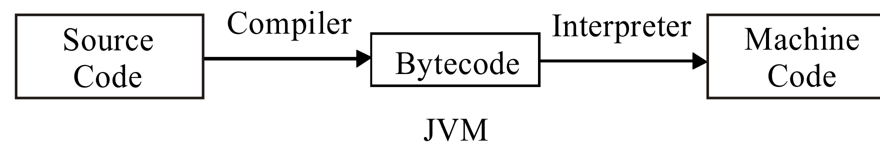


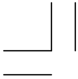
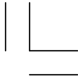
Fig. 4.5 *Execution of a Java Program*

Check Your Progress

8. Compare between Java and C++.
9. Define JVM.

4.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Java is an object oriented programming language that is based on the concept of an object.
2. Java programs are of two types—applications and applets.
    a. A Java application is a standalone program that can be invoked from the command line.
    b. A Java applet is a small program embedded in a Web page, and it will be run when that page is browsed using a Web browser.
3. Java interpreter will interpret and run the bytecode at runtime.
4. Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time.
5. Inheritance can be defined as the process whereby an object of a class acquires characteristics from the object of another class.
6. The main advantage of inheritance is reusability where the existing classes can be simply re-used in new software instead of writing a new code and new features can be added without altering or modifying the features of the existing class.
7. Dynamic binding is the process of linking a function call to the actual code of the function at runtime.
8. The features of Java that differentiate it from C++ are as follows:
    - Java does not support the multiple inheritance of classes directly.
    - The concept of multithreading is supported by Java.
    - The destructor function in Java is replaced by the `finalize` method.

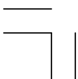
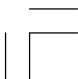



- The keyword `typdef` is not supported by Java.
- Java does not support pointers and instead it uses implicit object references.
- The virtual keyword is not supported in Java.
- Java does not support the concept of global variables.
- Java supports exception handling in a different way than C++. It provides the `finally` clause for cleanup.
- The non-primitive data types are allocated memory by using the new operator.
- Java adds many features that are necessary for object-oriented programming.

9. The source code written in Java is compiled to generate the bytecode and which is then interpreted to machine instructions for a specific machine. The bytecode generated by the compiler is not machine-specific. It is generated for the virtual machine that exists only inside the computer memory known as Java Virtual Machine (JVM).

4.8 SUMMARY

- Java is an object oriented programming language based on the concept of an object.
- Java is derived from C++, but is slightly simplified with libraries convenient for the Internet. It is the programming language for the Internet environment.
- The Internet implies heterogeneous systems, different network features, different windows libraries and different operating systems. Java guarantees identical program behavior on different platforms.
- Java is famous for its unique feature—platform independence (architecture neutral). This means that a Java program compiled on one machine could be ported to any other machine/operating system and executed without any modifications.
- Java is a highly object oriented language where reusability is of utmost importance. Java programs are very reliable on different platforms with the special features of memory allocation and de-allocation and exception handling.
- Java with its multithreaded approach can run many programs concurrently, thereby saving processor time. Synchronization of code is an added feature of Java to run non-erroneous interactive applications.
- In 1991, a group of engineers at Sun Microsystems Laboratory developed the Java language. The Green Project engineers designed a portable language

NOTES

that was small in nature, as its memory requirement was less and it can also be used to generate an intermediate code for virtual machines, such as Java Virtual Machine.

- The newly developed language was named Oak and was based on C++ and followed the object oriented programming approach. The Oak programming language was later renamed as Java and was introduced with this new name in 1995.
- The syntax for the Java language is similar to that of the C++ and C. The end user, who is familiar with C and C++, requires less effort in learning Java because Java and C or C++ have similar syntax.
- The Java environment comprises a set of tools and classes that are used to run the Java program.
- Java Development Kit contains various tools used by the Java Runtime Environment (JRE) that can be used to compile and interpret Java programs.
- Java standard library is one of the most attractive features of Java that contains various classes to support all the major functions of Java.
- Java uses both compiler and interpreter. The source code written in Java is compiled to generate bytecode and then this bytecode is interpreted to machine instructions for a specific machine.
- Objects are small, self-contained and modular units with a well defined boundary. An object consists of a state and behavior.
- A class is defined as a user defined data type which contains the entire set of similar data and the functions that the objects possess. In other words, a class in OOP represents a group of similar objects.
- A class serves as a blueprint or template for its objects. That is, once a class has been defined, any number of objects belonging to that class can be created. The objects of a class are also known as the instances or variables of that class and the process of creating objects from a class is known as instantiation.
- Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time. It allows the management of complex systems by concentrating on the essential features only.
- Encapsulation is the technique of binding or keeping the data and functions (that operate on them) together in a single unit called a class. Encapsulation is the way to implement data abstraction.
- Inheritance can be defined as the process where an object of a class acquires characteristics from the object of another class.
- Polymorphism is the ability of an entity, such as a function or a message to be processed in more than one form. It can also be defined as the property of an object belonging to a same or different class to respond to the same message or function in a different way.

- Message passing is the process of interacting between the different objects in a program.
- Dynamic binding is the process of linking a function call to the actual code of the function at runtime. That is, in dynamic binding, the actual code to be executed is not known to the compiler until runtime.
- Java is the latest all-purpose programming language. Its programs are collection of whitespace, identifiers, literals, comments, operators, separators and keywords.

4.9 KEY WORDS

- **Polymorphism:** It is the ability of an entity, such as a function or a message to be processed in more than one form.
- **Message passing:** It is the process of interacting between different objects in a program.
- **Dynamic binding:** It is the process of linking a function call to the actual code of the function at runtime.
- **Multithreaded:** Java with its multithreaded approach can run many programs concurrently, thereby saving processor time. Synchronization of code is an added feature of Java to run non-erroneous interactive applications.
- **Abstraction:** Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time. It allows the management of complex systems by concentrating on the essential features only.

4.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. List some of the important features of Java programming language.
2. List the components of Java Development Kit.
3. What is message passing? How is it used in object oriented programming?
4. List the advantages of object oriented programming.
5. What is Java run-time environment?

Long-Answer Questions

1. Write notes on the following:
    a. Abstraction

NOTES

b. Inheritance

c. Polymorphism

2. Explain the defining features of the Java programming language.
3. Outline the benefits of OOPS concept.
4. Discuss the evolution of Java.
5. Differentiate between Java and C++.

4.11 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

UNIT 5 OVERVIEW OF JAVA LANGUAGE

NOTES

Structure

5.0 INTRODUCTION

Java, initially named '*Oak*' was developed by a team headed by James Gosling at Sun Microsystems of USA in 1991. The primary reason behind the development of Java was the need of a platform independent and portable software to be embedded in the consumer's electronic devices like remote controls, microwave ovens, etc. Another reason that led to the growth of Java is the Internet and the World Wide Web as these media need portable and platform independent programs. Gosling and other team members developed web applets using the new language that could run on all types of computers. During 1993, the first web browser, namely 'HotJava', was developed to locate and run applet programs. This development made Java language popular for the Internet. By the year 1996, Java developed into a general-purpose, object-oriented programming language which was used for Internet programming. Soon, Java became popular and many web browsers like Internet Explorer, Netscape Navigator, etc., incorporated the ability to run Java applets.

NOTES

Java technology is both a programming language and a platform. Java is a high level, robust, secured and Object Oriented Programming (OOP) language. Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment, the Java Runtime Environment (JRE) and Application Programming Interface (API), it is called platform.

In this unit, you will study about the overview of Java language, simple Java program, comments, Java program structure, tokens, Java statements, implementing a Java program, JVM, command line arguments, constants, variables, data types and type casting.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the significant properties of Java language
- Analyse the Java program structure
- Write simple Java program
- Define the importance of comments and tokens in Java
- Use the Java statements in Java program
- Discuss the importance of JVM and command line arguments
- Use constants, variables and data types in the Java program
- Understand what type casting is

5.2 BASIC FEATURES OF JAVA PROGRAMMING LANGUAGE

Java is a third generation programming language which implements most of the concepts of OOPs. This section presents the basic features of Java as well as the characteristics that differ it from C++. It also describes the significance of Java to the Internet.

5.2.1 Features

Java has become a popular language for Internet applications because of various features which are as follows:

- **Easy to Use:** Java inherits the syntax of C/C++ and many of the OOPs features of C++; thus a person who has understood the concepts of object-oriented language can learn Java quite easily. Moreover, Java omits the complex and unreliable code of C and C++ like operator overloading,

pointers, preprocessor header files, etc. Java provides small and convenient ways to accomplish a given task.

- **Interpreted**: Unlike other languages, Java uses a two-stage system as it uses both compiler and interpreter for its program execution. The compiler converts the program code to a **bytecode,** which in turn is converted to a machine code on any machine using the interpreter. The machine code so generated can be executed irrespective of the system on which it is being executed.
- **Architecture-Neutral (Platform-Independent)**: This feature makes Java language very special. Java programs can run on any platform, i.e., they can run on different CPUs and on different operating system architectures. The bytecode produced by Java compiler can be run on any machine that has a Java runtime environment.
- **Object-Oriented Language**: Java is an object-oriented language as everything in Java is an object. The objects and classes contain the program code and data. The Java object model is easily extensible and classes can be used anywhere in the program in form of packages.
- **Robust**: Java is a robust language mainly because of two reasons, i.e., it is a strictly typed language that checks the code at the compile time and it uses an effective technique of memory management. In C++, the programmer has to manually deallocate the dynamic memory used by the objects but Java automatically deallocates free memory that is no longer referenced by the objects (with the help of garbage collector).
- **Distributed**: Since Java is platform independent, it is suitable for developing applications for the networks. Java can handle TCP/IP protocols and hence applications developed in Java can access remote objects on Internet like any object on a local system.
- **Multithreaded**: Java supports multithreaded programming which allows us to write a program that can perform more than one task simultaneously. The user is not required to wait for a particular program or to finish a task before starting the next task. For instance, a user can listen to an audio clip while downloading the applet. This feature helps to improve the performance of graphical applications.
- **High Performance**: The Java program is converted to a bytecode which is then converted to a machine code using the interpreter. Since bytecode is highly optimized, it enables the JVM to execute programs at a high speed.
- **Dynamic**: Java is dynamic in nature, i.e., Java programs can link to new class libraries, objects, methods, etc., at the run-time. Java language also provides the facility to include the functions of other languages like C and C++. They are referred to as **native methods**. These methods are also linked dynamically at run-time.

5.2.2 Java and C++

NOTES

Both C++ and Java are object-oriented languages; however, they are quite different from each other. Some of the features of C++ language were deliberately removed and some new features were added to make Java more flexible and reliable. Thus, there are various features of Java that differ it from C++. They are as follows:

- Java does not support multiple inheritance of classes directly.
- The concept of multithreading is supported by Java.
- The destructor function in Java is replaced by the finalize method.
- The keywords typdef is not supported by Java.
- Java does not support pointers, instead it uses implicit object references.
- The virtual keyword is not supported in Java.
- Java does not support the concept of global variables.
- Java supports exception handling in a different technique than C++. It provides the final clause for cleanup.
- The non-primitive data types are allocated memory by using the new operator.
- Java adds many features that are necessary for object-oriented programming.

5.2.3 Java's Importance in the Internet

In recent years, Java has become a popular language for the programs that are required to run on different systems. Java's new innovation named 'applet' has completely transformed Internet programming. Applets are tiny programs that are designed in such a way that they can be transmitted over the Internet. They can be downloaded on demand and executed automatically by a Java compatible web browser. They are used to handle user input, data supplied by the server and simple functions that execute locally on the client machine.

Applet is a dynamic, self-executing program and can be modified according to user inputs. The dynamic programs when downloaded and executed can cause serious damage to the computer as it may contain viruses like Trojan horse and other malicious programs. These harmful programs may search the contents on a local file system of the client computer and may gather private information like credit card numbers, passwords, etc. Earlier, viruses were scanned before executing the downloaded program, but Java has resolved the issue by confining Java programs to a Java execution environment only.

5.3 JAVA COMPILER, JAVA VIRTUAL MACHINE CONCEPTS AND JDK (JAVA DEVELOPMENT KIT)

NOTES

In most programming languages, the program is converted to machine code either by using the compiler or interpreter. The machine code so generated is machine dependent, i.e., it may not run on the machine other than the one on which it is generated. Unlike other programming languages, the Java compiler does not convert source code to machine code; rather it converts source code to a special intermediate code known as the **bytecode**. The bytecode so generated are in the form of class files that can be interpreted. The command used for compilation in Java is javac which converts the corresponding Java file into a class file. The bytecode is machine independent which implies that it can be run on any machine with the help of a Java virtual machine (JVM). Figure 5.1 illustrates the compilation of a Java program.

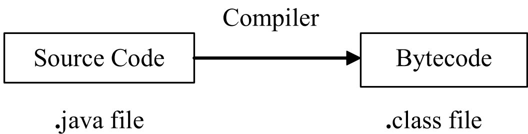


Fig. 5.1 *Compilation of a Java Program*

Java Virtual Machine Concepts

Java uses both compiler and interpreter. The source code written in Java is compiled to generate a bytecode and this bytecode is then interpreted to machine instructions for a specific machine. The bytecode generated by the compiler is not machine specific. It is generated for the virtual machine that exists only inside the computer memory known as the **Java Virtual Machine** (JVM). This virtual machine is designed in such a way that it can be implemented on the top of any existing processor and functions as a virtual processor chip. It hides the underlying operating system details from Java applications. Figure 5.2 illustrates the execution of a Java program.

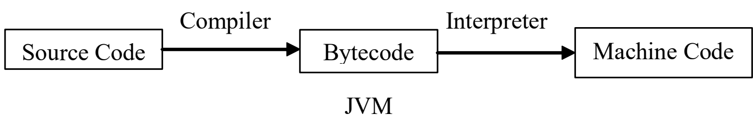


Fig. 5.2 *Execution of a Java Program*

JDK (Java Development Kit)

JDK consists of various tools that are used to develop and execute Java programs. The tools included in JDK are listed in Table 5.1.

NOTES

Table 5.1 Tools in Java Development Kit

Tool	Function
javac	Java compiler that converts source code to Java bytecode
java	Java interpreter that interprets class files generated by java compiler and converts it to machine code
javadoc	Document generator which automatically generates documentation form source code
javah	Generates C headers and a stub generator used for writing native methods
javap	The class file disassembler which enables to convert bytecode files to a program description
jdb	Java debugger which helps in tracking errors in the program
appletviewer	Used for running and debugging Java applets without a web browser

Check Your Progress

1. What is Java?
2. What is the function of the Java interpreter?
3. How is Java platform independent?
4. Distinguish between Java and C++.
5. Define JVM concept.

5.4 CHARACTER SET AND TOKENS

A **character set** can be defined as a set of characters that either individually or in combination, represents information. Unlike other programming languages like C and C++, Java uses a standard known as **unicode** to define characters. Unicode is a 16 bit character code set which defines all of the characters available in all human languages like English, Hindi, French, German, Chinese, Japanese, etc. This makes Java a worldwide programming language. However, the characters of many languages can be represented using the standard 8 bit character set (ASCII characters) which include letters, digits and punctuation marks.

A **token** is defined as the smallest unit of a program. When a program is compiled, the compiler scans the source code and parses it into tokens to find the syntax errors. Java tokens are broadly classified into *keywords*, *identifiers*, *constants*, *operators* and *punctuators*.

5.4.1 Java Keywords

Keywords are the predefined words that have special significance in any language. Every keyword is reserved for a specific purpose and hence cannot be used as user-defined names (identifiers). There are various keywords in Java, which are listed in Table 5.2.

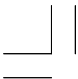



Table 5.2 *Java Keywords*

Keywords					
abstract	const	final	interface	short	transient
assert	continue	finally	long	static	try
boolean	default	for	native	strictfp	void
break	do	goto	new	super	volatile
byte	double	if	package	switch	while
case	else	instanceof	private	synchronized	
catch	enum	implements	protected	this	
char	extends	import	public	throw	
class	float	int	return	throws	

Notes*: The keywords* `const` *and* `goto` *are reserved words but they are not used.*

NOTES

5.4.2 Identifiers

Identifiers are the names given to uniquely identify various programming elements like *variables*, *arrays*, *methods*, *classes*, *objects*, *packages, interface*, and so on. While defining identifiers in Java, programmers must keep in mind the following rules:

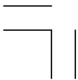
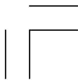
- An identifier must be unique in a program.
- Alphabets, digits, underscore and dollar sign characters can be used in an identifier.
- An identifier must not start with a digit.
- An identifier in upper case is different from that in lower case.
- An identifier must not contain other characters, such as '`*`', '`;`' and white space characters (tabs, space and newline).

Some valid and invalid identifiers in Java are as follows:

```
Pol78_ddm      //valid
hh$gl              //valid
_78hhvt4      //valid
902gt1        //invalid as it starts with a digit
Tyy;ui8    //invalid as it contains the ';' character
for        //invalid as it is a Java keyword
Fg026 neo  //invalid as it contains spaces
```

5.4.3 Constants

Constants, also known as **literals**, are the values that a program cannot alter during its execution. For instance, `391`, "`Byron`", `51.072` and '`p`' are all constants. Based on the type of value (data), Java constants are broadly classified into four categories, namely, numeric constants, character constants, string constants and boolean constants.

NOTES

Numeric Constants

Numeric constants refer to the numbers consisting of a sequence of digits (with or without decimal point) that can be either positive or negative. By default, numeric constants are positive. Numeric constants can be further classified as integer constants and floating-point constants which are listed in Table 5.3.

Table 5.3 *Type of Numeric Constants*

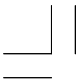
Type	Description	Example
Integer Constants	Integer constants refer to integer-valued numbers. Integer constants can be represented by three different number systems namely, *decimal* (base 10), *octal* (base 8) and *hexadecimal* numbers (base 16). The octal constants are preceded by a 0 (zero) and hexadecimal constants are preceded by a 0x or 0X.	54, -646, 01612, 0x38A
Floating Point Constants	Floating-point constants refer to the real numbers, that is, the numbers with a decimal point. Floating-point constants are also written in the *floating-point notation* in which the constant is divided into a *mantissa* and an *exponent.* Floating point constants default to double precision, so we should append an `F` or `f` to the constant in order to specify a float constant.	64.23f, -74.32f, 537E-9f, 15e-3f

Note: *Use of special characters like comma ',', semicolon ';'and question mark '?' are not permitted in numeric constants.*

Character Constants

Character constants refer to a single character enclosed in single quotes (''). Character constants include '`f`', '`M`', '`8`', '`&`', etc. All character constants are internally stored as integer value.

Character constants can represent either printable characters or non-printable characters. Printable character constants include '`a`', '`5`', '`#`', '`;`', etc. However, there are a few character constants that cannot be included in a program directly through a keyboard, such as backspace, new line, and so on. These character constants are known as **non-printable constants** and are included in a program using the *escape sequences*. An escape sequence refers to a character preceded by the backslash character (\). Some of the escape sequences used in Java are listed in Table 5.4.




NOTES

Table 5.4 *Escape Sequences*

Escape Sequences	Character Constants
`\b`	Backspace
`\f`	Form feed
`\n`	Newline (Linefeed)
`\r`	Carriage return
`\t`	Tab
`\'`	Single quote
`\"`	Double quote
`\\`	Backslash

String Constants

String constants refer to a sequence of any number of characters enclosed in double quotes (""). The characters can be alphabets, digits, special characters and blank spaces. String constants include "hello java", "2009", "!...?", "6+9", "Y", etc.

Boolean Constants

Boolean constants can represent only two values, true and false. In Java, there is no numerical representation for these values, i.e., true is not equal to 1 and false is not equal to 0.

Operators

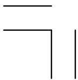
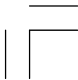
Operators are symbols which perform operations, (such as addition, subtraction, multiplication, etc.), on various data items to produce a result. These data items on which operators act are known as **operands**. For instance, in a*b, a and b are operands and * is an operator.

Punctuators

Punctuators, also known as **separators**, are the symbols that define the structure of a program by dividing and arranging a set of codes. The various punctuators defined are braces '{ }', brackets '[ ]', colon ':', comma ',', period '.', semicolon ';' and parentheses '( )'.

5.5 STRUCTURE OF A JAVA PROGRAM

The Java program structure is divided into various sections, namely, documentation section, package statement, import statements, interface statements, class definitions and main method class as displayed in Figure 5.3.

NOTES

Documentation Section
Package Statement
Import Statements
Interface Statements
Class Definitions
Main Method Class

Fig. 5.3 **Java Program Structure**

The various sections of a Java program are described as follows:

- **Documentation Section**: Comments are vital elements of a program that are used to increase its readability and describe its functionality. They may also include information like program name and the author of the program, etc.
- **Package Statement**: It is the first statement in Java program that tells the compiler that all the classes defined in the file belong to this package. For instance, consider the following statement:

```
package employee;
```

  Here, `employee` is the name of the package. It is not necessary that our classes are part of the package, so this statement is optional.
- **Import Statements**: It allows us to access a class, which belongs to some other package. For instance, consider the following statement:

```
import employee.EName;
```

  Here, `employee` is the name of the package and `EName` is the class which we want to access. There can be a number of import statements in a program.
- **Interface Statements**: An interface is a way for implementing multiple inheritance in Java. It is just like a class but contains only method declarations.
- **Class Definitions**: Class is the most important element of a program. A program may consist of any number of class definitions.
- **`Main` Method Class**: It is an essential part of a Java program as it contains the `main` method which is the starting point of a program. Inside the `main` method class, the objects of several classes can be created, accessed and manipulated. Once all the instructions in the main method are executed, the control is transferred out of the class, thus terminating the entire program.

Note: *The concept of package, interface and multiple Inheritance are discussed in subsequent chapters.*

5.6 DATA TYPES

NOTES

A data type determines the type and the operations that can be performed on the data. Java provides various data types and each data type is represented differently within the computer's memory. The type of data selected by a programmer depends on the particular application. The various data types provided by Java are categorized into primitive data types and non-primitive data types as displayed in Figure 5.4.

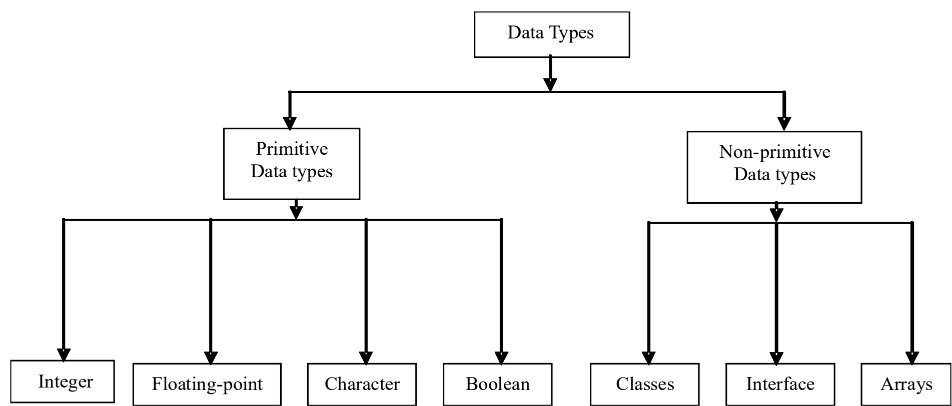


Fig. 5.4 *Data Types*

5.6.1 Primitives Data Types

Primitive data types, also known as **built-in data types** are the fundamental data types provided by a programming language. In Java, primitive data types include integer, floating point, character and boolean.

Integer Type

The integer data type is used to store integers like `4`, `42`, `5233`, `-32`, `-745`. Java supports four types of integers, namely, `byte`, `short`, `int` and `long`. The default value of these integer types is 0. There is no concept of unsigned integer in Java. The various integer data types with their size and range are listed in Table 5.5.

Table 5.5 *Size and Range of Integer Types*

Type	Size(bytes)	Range
byte	One	-128 to 127
short	Two	-32,768 to 32,767
int	Four	-2,147,483,648 to 2,147,483,647
long	Eight	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Note: *It is recommended to use smaller data types whenever possible. This is because larger the data type we choose, more time the program will take for execution.*

NOTES

Floating Point Type

A floating point data type is used to store real numbers, such as `3.28`, `64.755765`, `-8.01`, `-24.53`. Java supports two floating point data types, namely, `float` and `double`.

- **`float`**: The `float` type represents a *single-precision* number. Single precision occupies lesser space than double precision but becomes inaccurate when the values are large. For instance, it can be used to represent the value of the marks obtained by students. The default value of `float` data type is `0.0f`.
- **`double`**: The `double` type specifies a *double-precision number*. It is the best choice when we need to store large-valued numbers. For instance, it can be used when we want to use mathematical functions like `sin()`, `cos()`, `sqrt()`. The default value of `double` data type is `0.0d`.

The various floating point data types with their size and range are listed in Table 5.6.

Table 5.6 *Size and Range of Floating Point Types*

Type	Size(bytes)	Range
float	Four	3.4e-038 to 3.4e+038
double	Eight	1.7e-308 to 1.7e+308

Character Type

The character data type is used to store single character enclosed in single quotes**.** It is represented by using the `char` keyword. It occupies 16 bits of memory. The range of the character data type is 0 to 65,536. The default value of `char` data type is a null character.

Boolean Type

The boolean data type can hold only boolean values, i.e., either `true` or `false`. The keyword `boolean` is used to denote the boolean data type. The default value of `boolean` data type is `false`.

Non-Primitive Data Types

Non-primitive data types (user-defined data types) also known as **reference types** are derived from the primitive data types. In Java, these include *classes*, interface and arrays.

5.7 VARIABLES

A variable is an identifier that represents a memory location that is used to store data value. Data stored at a particular location can be accessed using the variable

name. The value of a variable can be changed anytime during the program execution. The variable name we choose must be meaningful so as to understand what it represents in the program.

Declaring Variables

Variables must be declared in a program before they are used. The declaration of a variable informs the compiler, the specific data type to which a variable is associated and allocates sufficient memory for it.

The syntax for declaring a variable is as follows:

```
data_type variable_name;
```

For instance, a variable `a` of type `int` can be declared using the following statement:

```
int a;
```

At the time of the variable declaration, more than one variable of the same data type can be declared in a single statement as in the case of the following statement:

```
int x, y, z;
```

Initializing Variables

Declaration of variables allocates memory for variables but it does not store any data at the time of declaration. To store data in the variables, they need to be initialized. For instance, consider the following statements:

```
int i;
i=10;
```

Here, a variable `i` of the integer type is declared and the value 10 is assigned to it. We can combine both the statements into a single statement as follows:

```
int i=10;
```

Besides initializing the variable with the constant values, variables can also be initialized at run-time using expressions. Initialization of variables at run-time is known as **dynamic initialization**.

Program 1: A program to demonstrate initialization of a variable.

```
public class dynamic_initializaton
{
      public static void main(String[] args)
     {
             int x=40,y=40,z=10; //initialization with constant
                   values
          int result=(x*y)+z; //dynamic initialization
          System.out.println("The value of z is:"+result);
     }
}
```

Output of the program:

```
The value of z is:1610
```

Receiving Input Through Keyboard

Variables can also be given values interactively through the keyboard using the `readLine()` method.

NOTES

Program 2: A program to demonstrate reading data from the keyboard.

```
//importing package for using DataInputStream class
import java.io.*;
public class ReadingData
{
     public static void main(String[] args)
     {
          DataInputStream in=new DataInputStream(System.in);
          int num1=0;
          float num2=0;
          try
          {
               System.out.println("Enter integer value");
               num1=Integer.parseInt(in.readLine());
          System.out.println("Enter float value");

num2=Float.valueOf(in.readLine()).floatValue();
          }
           catch(Exception e)
          {
          }
          System.out.println("The integer value is "+num1);
          System.out.println("The float value is "+num2);
     }
}
```

Output of the program:

```
Enter integer value
4
Enter float value
6.7
The integer value is 4
The float value is 6.7
```

The method `readLine()` of class `DataInputStream` is used to read string from the keyboard which is then converted to the corresponding data type, `int` and `float`. To handle the error which may occur while reading data from the keyboard, we have provided `try` and `catch` statements.

Check Your Progress

6. State the difference between integer constants and floating point constants.
7. What is the structure of the Java program?
8. Discuss the characteristics of import statements in the Java program.
9. Classify various types of data types.
10. How do you initialize variable in a Java program?

NOTES

5.8 JAVA PROGRAM

Let us start learning Java with a simple Java program that prints a string on the screen.

Program 3: A Java program.

```
class Sample
{
    public static void main (String args[])
    {
       System.out.println("Welcome to Java Programming");
       System.out.println("Its easy and simple.");
    }
}
```

Though this program is the simplest one, it includes the basic features that every Java program has. Let us now look at the features one by one.

Class Definition

The first statement class Sample declares a class where class is a keyword and Sample is the identifier that indicates the name of the class. The opening and closing curly braces '{ }' enclose the definition of a class.

The Main Statement

The statement `public static void main (String args[])` indicates the `main` method. This method is the point where the execution of Java program begins. Since it is the startup point of any Java program, it is the most essential part of any Java program.

This statement has keywords, namely, `public`, `static` and `void`. The descriptions of these keywords are as follows:

- `Public`: It is access specifier which specifies that the `main` method is accessible to all other classes.
- `Static`: The `main` method is declared static which specifies that this method belongs to the entire class. The interpreter use this method before the creation of objects.
- `Void`: The `void` keyword specifies that the `main` method does not return a value. The pair of parentheses contains the declaration of the parameters of the methods. In the above statement, the `String args[ ]` declares a parameter `args`, that contains an array of objects of the class type `String`.

NOTES

The Output Statement

The statements `System.out.println("Welcome to Java Programming.");` and `System.out.println("Its easy and simple.");` are used to display information on the standard output device, i.e., the `println() which` is a method of the `out` object, a static member of the class `System`. These statements will display the following strings on the monitor:

```
Welcome to Java Programming.
Its easy and simple.
```

The statements will be printed in separate lines as the method `println()` appends a newline character at the end of the string. However, if we use the `print()` method instead of the `println(), the` newline character is not appended at the end of the string. Also, like C++, every statement in Java must end with a semicolon.

5.9 RUNNING JAVA APPLICATIONS AND COMMAND LINE ARGUMENTS

A program can be created using any text editor. Nowadays, there are several text editors available for writing programs like Notepad, Jcreator, etc. After creating the file, save the file with the name `<filename>.Java`. The name of the file must be same as that of the class name containing the `main()` method.

For instance, once the file is created, it can be compiled to generate the bytecode using Java compiler `javac` as follows:

javac Sample.java

If the source code is error free then the compiler creates a file containing the bytecode and the file will be named as `<filename>.class`. Here, the name of file will be `Sample.class`.

Even though after compilation the source code is converted into its equivalent bytecode, it cannot be executed. To execute this bytecode, it needs to be converted to a machine code using the interpreter. The command for converting the bytecode to the machine code and run it is as follows:

```
java Sample
```

After giving this command, the interpreter searches for the `main()` method in the source code and starts executing the instructions written in this method displaying the corresponding output.

Command Line Arguments

Like C++, Java enables to pass arguments to the `main()` method also. These arguments are passed by typing them after the program name on the command

line. Hence, these arguments are known as **command line arguments**. They help in providing data to the program.

The argument named as `args` of `String` type is passed to the `main` method. It is this argument which will receive any data passed to the program through command line.

Program 4: A program to demonstrate the use of command line arguments.

```
class CommandLineArg
{
    public static void main(String args[])
    {
System.out.println("The arguments entered are: ");
int a;
for (a=0;a<args.length;a++)
{
System.out.print("args[" + a + "]: ");
System.out.println(args[a]);
}
int b=args.length; // returns the number of arguments
System.out.println("The number of arguments is: "+b);
}
}
```

The command entered on the command prompt is as follows:

```
java CommandLineArg Hello! How are you?
```

Output of the program:

```
The arguments entered are:
args[0]: Hello!
args[1]: How
args[2]: are
args[3]: you?
The number of arguments is: 4
```

5.10 TYPE CASTING

While programming we might come across some situations where a value of one data type needs to be stored into a variable of another data type. If both the data types are compatible and the data type of the target variable is large enough to store the value of the source variable, Java automatically converts the source type into target type. This is known as **automatic type conversion**. For instance, the value of `int` data type can be assigned to a variable of `long` data type since `long` is larger than `int`. This type of conversion in which data of smaller type is assigned to a larger type is called **widening conversion**.

NOTES

However, if the target type is smaller than the source type, conversion cannot be performed automatically. For instance, the value of `int` data type cannot be assigned to a variable of `byte` type. For such conversion, Java provides a mechanism known as **type casting**. Type casting refers to the type conversion that is performed explicitly.

The syntax for type casting is as follows:

```
data_type variable1 = (data_type) variable2;
```

where,

`data_type` is the data type

`variable1` is the target variable

`variable2` is the source variable

For instance, consider the following statements:

```
int x = 20;
byte y = (byte) x;
```

In these statements, type casting is performed to convert `int` data type to `byte` data type.

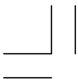
The following points should be kept in mind while type casting:

- All the integer data types can be cast to any other data type except `boolean`.
- Casting into smaller data type may lead to loss of data.
- Casting a floating point value to an integer type may result in truncation of the fractional part.

Program 5: A program to demonstrate type casting.

```
class ExampleTypecast
{
      public static void main(String[] args)
      {
            byte b = 20;
            short s = 26;
            int i = 123456789;
            float f = 3.98f;
            System.out.println("The existing variables and
            their values are:");
            System.out.println("b="+b);
            System.out.println("s="+s);
            System.out.println("i="+i);
            System.out.println("f="+f);

            System.out.println("The existing variables and
            their values after conversion are:");
            System.out.println(" ");
   // type conversion
            int i1=(int)b;  // conversion from byte to int
            short s1=(short)b; // conversion from byte to
      short
```





```
            short s2=(short)i; // conversion from int to short
            int i2 =(int)f;  // conversion from float to int

          System.out.println("(int)b: "+i1);
          System.out.println("(short)b: "+s1);
            System.out.println("(short)i: "+s2);
            System.out.println("(int)f: "+i2);
      }
}
```

NOTES

Output of the program:

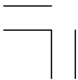
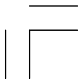
```
The existing variables and their values are:
b=20
s=26
i=123456789
f=3.98
The existing variables and their values after conversion are:
(int)b: 20
(short)b: 20
(short)i: -13035
(int)f: 3
```

Check Your Progress

11. Write a program to print a string on the screen.
12. Define widening conversion.
13. Write a syntax for type casting in a Java program.

5.11 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Java is a third generation programming language which implements most of the concepts of OOPs.
2. The compiler converts the program code to a bytecode, which in turn is converted to a machine code on any machine using the interpreter.
3. Java programs can run on any platform, i.e., they can run on different CPUs and on different operating system architectures. The bytecode produced by Java compiler can be run on any machine that has a Java runtime environment.
4. There are various features of Java that differ it from C++. They are as follows:
   - Java does not support multiple inheritance of classes directly.

NOTES

- The concept of multithreading is supported by Java.
- The destructor function in Java is replaced by the finalize method.
- The keywords typdef is not supported by Java.
- Java does not support pointers, instead it uses implicit object references.
- The virtual keyword is not supported in Java.
- Java does not support the concept of global variables.
- Java supports exception handling in a different technique than C++. It provides the final clause for cleanup.
- The non-primitive data types are allocated memory by using the new operator.
- Java adds many features that are necessary for object-oriented programming.

5. Java uses both compiler and interpreter. The source code written in Java is compiled to generate a bytecode and this bytecode is then interpreted to machine instructions for a specific machine. The bytecode generated by the compiler is not machine specific. It is generated for the virtual machine that exists only inside the computer memory known as the Java Virtual Machine (JVM). This virtual machine is designed in such a way that it can be implemented on the top of any existing processor and functions as a virtual processor chip. It hides the underlying operating system details from Java applications.

6.

Type	Description	Example
Integer Constants	Integer constants refer to integer-valued numbers. Integer constants can be represented by three different number systems namely, *decimal* (base 10), *octal* (base 8) and *hexadecimal* numbers (base 16). The octal constants are preceded by a 0 (zero) and hexadecimal constants are preceded by a 0x or 0X.	54, -646, 01612, 0x38A
Floating Point Constants	Floating-point constants refer to the real numbers, that is, the numbers with a decimal point. Floating-point constants are also written in the *floating-point notation* in which the constant is divided into a *mantissa* and an *exponent.* Floating point constants default to double precision, so we should append an `F` or `f` to the constant in order to specify a float constant.	64.23f, -74.32f, 537E-9f, 15e-3f

7. The Java program structure is divided into various sections, namely, documentation section, package statement, import statements, interface statements, class definitions and main method class.

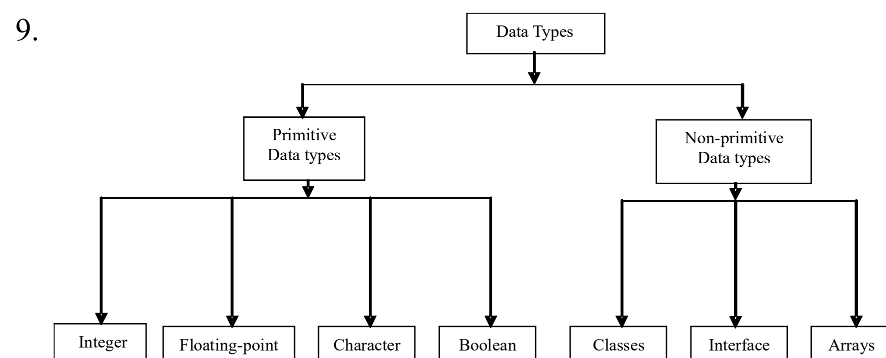
NOTES

8. Import statements: It allows us to access a class, which belongs to some other package. For instance, consider the following statement:

```
import employee.EName;
```

Here, `employee` is the name of the package and `EName` is the class which we want to access. There can be a number of import statements in a program.

9.



10. Declaration of variables allocates memory for variables but it does not store any data at the time of declaration. To store data in the variables, they need to be initialized. For instance, consider the following statements:

```
int i;
i=10;
```

Here, a variable `i` of the integer type is declared and the value 10 is assigned to it. We can combine both the statements into a single statement as follows:

```
int i=10;
```

Besides initializing the variable with the constant values, variables can also be initialized at run-time using expressions. Initialization of variables at run-time is known as **dynamic initialization**.

11. Java program that prints a string on the screen.

A Java program.

```
class Sample
{
public static void main (String args[])
{
System.out.println("Welcome to Java Programming");
System.out.println("Its easy and simple.");
}
}
```

NOTES

12. For instance, the value of `int` data type can be assigned to a variable of long data type since long is larger than `int`. This type of conversion in which data of smaller type is assigned to a larger type is called widening conversion.
13. The syntax for type casting is as follows:

```
data_type variable1 = (data_type) variable2;
```

where,

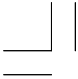
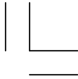
`data_type` is the data type

`variable1` is the target variable

`variable2` is the source variable

5.12 SUMMARY

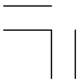
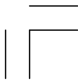
- Java uses two-stage system as it uses both compiler and interpreter for its program execution.
- Applets are tiny programs that are designed in such a way that they can be transmitted over the Internet.
- Java compiler does not convert source code to machine code, it converts source code to a special intermediate code known as bytecode. The command used for compilation in Java is `javac` which converts the corresponding Java file into class file.
- Bytecode is generated for the virtual machine that exists only inside the computer memory known as Java Virtual Machine (JVM).
- JDK consists of various tools that are used to develop and execute Java programs. Some of the tools included in JDK are javac, java, javadoc, javap, etc.
- The access specifier `public` specifies that the `main` method is accessible to all other classes.
- The `void` keyword specifies that the `main` method does not return a value.
- A character set can be defined as a set of characters that either individually or in combination, represents information. Unlike C and C++, Java uses a standard known as Unicode to define characters.
- Unicode is a 16 bit character code set which defines all of the characters available in all human languages.
- A token is defined as the smallest unit of a program. Java tokens are broadly classified into keywords, identifiers, constants, operators and punctuators.
- Keywords are the predefined words that have special significance in any language. Every keyword is reserved for a specific purpose and hence cannot be used as user-defined names (identifiers).

- The Java program structure is divided into various sections, namely, documentation section, package statement, import statements, interface statements, class definitions and main method class.
- A data type determines the type and the operations that can be performed on the data. The various data types provided by Java are categorized into primitive data types and non-primitive data types. Primitive data types also known as built-in data types are the fundamental data types provided by a programming language. In Java, primitive data types include integer, floating point, character and boolean.
- Java supports four types of integer, namely, `byte`, `short`, `int` and `long`. It supports two floating point data types, namely, `float` and `double`.
- A variable is an identifier that represents a memory location that is used to store data value. The value of a variable can be changed anytime during the program execution.

5.13 KEY WORDS

- **Character set:** A set of characters that either individually or in combination, represents information.
- **Unicode:** A16 bit character code set which defines all of the characters available in all human languages.
- **Token:** The smallest unit of a program.
- **Identifiers:** Names given to uniquely identify various programming elements including variables, arrays, methods, classes, objects, packages, interface, etc.
- **Constants**: The values that a program cannot alter during its execution.
- **Numeric constants:** The numbers consisting of a sequence of digits (with or without decimal point) that can be either positive or negative.
- **Character constants:** A single character enclosed in single quotes (‘’). Character constants include ‘`f`’, ‘`M`’, ‘`8`’, ‘`&`’, etc.
- **String constants:** A sequence of any number of characters enclosed in double quotes (“”).
- **Boolean constants:** Constants that can represent only two values, `true` and `false`.
- **Punctuators:** The symbols that define the structure of a program by dividing and arranging a set of codes.
- **Dynamic initialization:** Initialization of variables at run-time.

5.14 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. Mention some of the differences between Java and C++.
2. What is the role of Java compiler in the execution of the program?
3. Describe a Java program structure.
4. What are the various primitive data types in Java?
5. What is type casting? When do we need it in programming?
6. What are the different ways of initializing a variable? Explain with an example.
7. What are command line arguments? Explain with an example.
8. State the difference between the execution of `while` and `do-while` loop. Support your answer with suitable examples.
9. Describe the purpose of JDK. Mention some tools of JDK and their purpose.
10. Define an array. Why are they needed?

Long-Answer Questions

1. Explain Java features in detail.
2. Explain how Java has become a popular language for the Internet.
3. Discuss Java runtime environment in detail.
4. Define token. Explain the various types of tokens supported by Java.
5. Write a program to display 'Java is a popular language for Internet' in Java.
6. Write a program to read two integer numbers from the keyboard and display their sum on the screen.

5.15 FURTHER READING

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

UNIT 6 OPERATORS AND EXPRESSIONS, DECISION-MAKING AND BRANCHING

Structure

6.0 INTRODUCTION

Expressions are constructed from operands and operators. The operators of an expression indicate which operations to apply to the operands. The order of evaluation of operators in an expression is determined by the precedence and associativity of the operators. An operator is a special symbol which indicates a certain process is carried out. Operators in programming languages are taken from mathematics. Because the programmers work with data, hence the operators are used for the processing of data. An operand is one of the inputs (arguments) of an operator.

In Java, the control statements are broadly classified into three categories, namely, conditional statements, iteration statements and jump statements. All these control statements are commonly used with the logical tests or test conditions to alter the flow of control conditionally or unconditionally. Decision-making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

In this unit, you will study about the operators and expressions, arithmetic operators, arithmetic expressions, evaluation of expression, precedence of arithmetic operators, type conversions, operator precedence and associativity, decision-making and branching, looping and labeled loops.

NOTES

6.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the significance of operators and expressions in Java language
- Explain the different types of Java operators
- Define the importance of arithmetic operators in Java
- Elaborate on arithmetic expressions, evaluation of expression and precedence of arithmetic operators in Java
- Use the arithmetic operators in Java programs
- Discuss the importance of type conversions, and operator precedence and associativity
- Understand the importance of decision-making and branching
- Elaborate on looping and labeled loops

6.2 OPERATORS AND EXPRESSIONS

Operators are the symbols which perform operations on various data items known as operands. For instance, in `a + b`, `a` and `b` are operands and + is an operator. Note that to perform an operation, operators and operands are combined together forming an ***expression***. For instance, to perform an addition operation on operands `a` and `b`, the addition (+) operator is combined with the operands `a` and `b` forming an expression.

Depending on the function performed, the Java operators can be classified into various categories. These include arithmetic operators, increment and decrement operators, relational operators, logical operators, conditional operators, assignment operators, bitwise operators and special operators.

Arithmetic Operators

Arithmetic operators perform the basic arithmetic operations on operands. They can work on any built-in data type of Java except on boolean type.

Java provides various arithmetic operators including +(addition or unary plus), –(subtraction or unary minus), *(multiplication), /(division) and %(modulus). For instance, some of the expressions which involve arithmetic operators are `x + y`, `x - y`, `x * y`, `x / y` and `x % y`. When the unary minus operator is used with a single operand, the operand is multiplied by `-1`.

Expressions formed by using arithmetic operators can be of the following types:

- **Integer Expression**: The arithmetic expression where both the operands are integers is called an integer expression.

- **Real Expression**: The arithmetic expression where both the operands are real is called real expression.
- **Mixed Mode Expression**: The expression is mixed mode if one operand is real and the other is an integer. In this case, the integer operand is converted to real and the result is also of type real.

Note: *Unlike C and C++, the modulus operator can be applied to the floating point data type also in Java.*

Increment and Decrement Operators

Java provides two special unary arithmetic operators, namely the increment operator (represented by ++) and the decrement operator (represented by –). The operator ++ increases the value of operand by 1 and the operator – decreases the value of operand by 1.

The increment and decrement operators can be used in the following forms:

- **Prefix Form**: In this form, the increment or the decrement operator *precedes* its operand. The prefix increment operator is represented as `++operand` and the prefix decrement operator is represented as `–operand`. The prefix increment or the prefix decrement operator increments or decrements the value of an operand respectively before its value is used in an expression.

  To understand prefix operators, consider the following illustration.

Program 1: Evaluate the following statements:

```
x=10;
y=++x;
z=–x;
```

In this illustration, the statement `y=++x` first increments the value of `x` by 1 and then assigns the incremented value to `y`. Similarly, the decrement operator is used to decrement the value of `x` by 1 and then assign the decremented value to `z`. Thus, the value of `y` and `z` are 11 and 10, respectively.

- **Postfix Form:** In this form, the increment or the decrement operator *succeeds* its operand. The postfix increment operator is represented as `operand++`, and the postfix decrement operator is represented as `operand–`. The postfix increment or the decrement operator increments or decrements the value of an operand, respectively, after using it in the expression.

To understand postfix operators, consider the following illustration.

Program 2: Evaluate the following statements:

```
x=10;
y=x++;
z=x–;
```

In this illustration, the statement `y=x++` first assigns the value of `x` to `y` and then increments the value of `x` by 1. Similarly, the statement `z=x–` assigns the value of `x` to `z` and then decrements the value of `x` by 1. Thus, the value of `y` and `z` are 10 and 11, respectively.

Relational Operators

Relational operators are used for comparing two values or expressions. The various relational operators provided by Java include less than '<', less than or equal to '<=', greater than '>', greater than or equal to '>=', equal to '==' and not equal to '!=' operator. They return values of boolean type, i.e., either `true` or `false`. For instance, consider two variables `a` and `b` having values 20 and 30 respectively. In this case, the expression `a<b` returns `true` whereas the expression `a>b` returns `false`.

The operators == and != are also known as *equality operators* as they are used for checking the equality of operands.

***Note**: All relational operators can work on integer, floating-point and character data types.*

Logical Operators

Logical operators combine expressions and then return `true` or `false`. The various logical operators provided by Java are as follows:

- **AND (&&) Operator**: It returns `true` only if all the expressions evaluate to `true`, otherwise it returns `false`.
- **OR (||) Operator**: It returns `true` if any one or all the expressions evaluate to `true` and returns `false` only if all the expressions evaluate to `false`.
- **Negation (!) Operator:** It returns `true` if the expression on which it is operating is `false` and vice versa.

For instance, consider two expressions `Exp1` and `Exp2`. Table 6.1 displays the result after the logical operators are applied on these expressions.

Table 6.1 *Truth Table*

Exp1	Exp2	Exp1 && Exp2	Exp1 OR Exp2	!Exp1
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Conditional Operators

The conditional operator selects a value based on a specified condition. Note that the conditional operator is a ternary operator, i.e., this operator involves three operands.

The syntax of the conditional operator is as follows:

```
expression1 ? expression2: expression3
```

If `expression1` is `true`, then `expression2` is evaluated, otherwise `expression3` is evaluated.

To understand the working of a conditional operator, consider the following illustration:

Program 3: Evaluate the following statement:

```
(x==5) ? 8 : 9;
```

In this illustration, if the value of x is equal to 5, then the expression returns 8, otherwise the expression returns 9.

Assignment Operators

Assignment operator assigns the value of an expression to a variable. Assignment operators are of two types, namely, the simple assignment operator and compound assignment operators.

Simple Assignment Operator

The simple assignment operator assigns the value on its right hand side to the variable on its left hand side. Note that the left hand side of an assignment expression should be a variable. It cannot be a constant or an expression. However, the right hand side of an assignment expression can be a variable, constant or an expression.

To understand the simple assignment operator, consider the following illustration.

Program 4: Evaluate the following statement:

```
x=8;
```

In this example, the value 8 is assigned to the variable x.

With the help of the assignment operator, several variables can be assigned a common value. This is accomplished by using multiple assignments in a single statement. For instance, in the statement x=y=z=5, the value 5 is assigned to the three variable x, y and z.

Compound Assignment Operators

Java provides compound assignment operators (also known as **Java shorthands).** These are in the following form:

```
v op=exp;
```

Here, v is a variable, op is the binary operator and exp is an expression. This form is equivalent to the statement v=v op(exp);
where we need to access v only once.

For instance, the expression x=x+6 can be written as x+=6. In this expression, x is incremented by 6 and then the result is assigned to x. The various compound assignment operators used in Java are '+=', '-=', '*=', '/=' and '%='.

Other Operators

In addition to the operators discussed, Java supports some other operators which include **bitwise** operators and **special** operators.

Bitwise Operators

The bitwise operators are used to manipulate the data values at bit level. This operator can be applied to all the primitive data types, such as long, int,

NOTES

`short`, `char` and `byte`. Various bitwise operators include *bitwise AND* (`&`), *bitwise OR* (`!`), *bitwise exclusive OR* (`^`), *one's complement* (`~`), *shift left* (<<), *shift right* (>>), *shift right with zero fill* (>>>).

Special Operators

Java provides two special operators, namely *`instanceof`* and *dot operator* (`.`).

Instanceof Operator

The `instanceof` operator is an operator which is used to check whether the object belongs to a particular class or not. For instance, consider the following statement:

```
novel instanceof book
```

This statement returns `true` if the object `novel` belongs to the class `book`, otherwise, it returns `false`.

Dot Operator

The instance variables and the methods of a class are accessed through objects with the help of dot operator. Dot operator links the name of the object with the name of the variable or method which needs to be accessed. For instance, consider the following statements:

```
obj.name;  // accessing instance variable of the class
obj.marks(); // accessing method of the class
```

Operators Precedence and Associativity

An expression consisting of more than one operator leads to a confusion as to which operator is to be evaluated first. For instance, consider the following expression:

```
a + b * c - d
```

In this expression, the compiler needs to know which operator is evaluated first. For this, it is important to determine the precedence and associativity of operators.

- **Precedence**: The order or priority in which various operators in an expression are evaluated is known as **precedence**. Every operator in Java has a precedence associated with it. The operators with a higher precedence are evaluated before the operators with a lower precedence. For instance, multiplication is performed before addition as the multiplication operator has higher precedence than the addition operator.
- **Associativity**: The order or priority in which operators of the same precedence are evaluated is known as **associativity**. For instance, the addition and subtraction operators have the same precedence. However, addition or subtraction may be performed on the expression depending upon the order of their occurrence.

The associativity of an operator can be either from left to right or from right to left. The operators with left to right associativity are evaluated from the left hand side while the operators with right to left associativity are evaluated from the right hand side.

The precedence and the associativity of Java operators are listed in Table 6.2. Note that the precedence of operators decreases from top to bottom, i.e., the priority is highest at the top.

Table 6.2 *Precedence and Associativity of Java Operators*

Operators	Description	Associativity
.	Direct member selector	Left to right
()	Function Call	Left to right
[]	Array subscript	Left to right
-	Unary minus	Right to left
++	Increment	Right to left
--	Decrement	Right to left
!	Logical negation	Right to left
~	Ones complement	Right to left
(type)	Casting	Right to left
*	Multiplication	Left to right
/	Division	Left to right
%	Modulus	Left to right
+	Addition	Left to right
-	Subtraction	Left to right
<<	Left shift	Left to right
>>	Right shift	Left to right
>>>	Right shift with zero fill	Left to right
<	Less than	Left to right
<=	Less than or equal to	Left to right
>	Greater than	Left to right
>=	Greater than or equal to	Left to right
instanceof	Type comparison	Left to right
= =	Equal to	Left to right
=!	Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise XOR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional operator	Right to left
=	Assignment Operator	Right to left
Op=	Shorthand assignment	

Note: *The operators in the same row have same precedence.*

NOTES

6.3 JAVA PROGRAM

Let us start learning Java with a simple Java program that prints a string on the screen.

Program 5: A Java program.

```
class Sample
{
    public static void main (String args[])
    {
       System.out.println("Welcome to Java Programming");
       System.out.println("Its easy and simple.");
    }
}
```

Though this program is the simplest one, it includes the basic features that every Java program has. Let us now look at the features one by one.

Class definition

The first statement class Sample declares a class where class is a keyword and Sample is the identifier that indicates the name of the class. The opening and closing curly braces '{ }' enclose the definition of a class.

The main statement

The statement `public static void main (String args[])` indicates the `main` method. This method is the point where the execution of Java program begins. Since it is the startup point of any Java program, it is the most essential part of any Java program.

This statement has keywords, namely, `public`, `static` and `void`. The descriptions of these keywords are as follows:

- `Public`: It is access specifier which specifies that the `main` method is accessible to all other classes.
- `Static`: The `main` method is declared static which specifies that this method belongs to the entire class. The interpreter use this method before the creation of objects.
- `Void`: The `void` keyword specifies that the `main` method does not return a value. The pair of parentheses contains the declaration of the parameters of the methods. In the above statement, the `String args[ ]` declares a parameter `args`, that contains an array of objects of the class type `String`.

NOTES

The Output Statement

The statements `System.out.println("Welcome to Java Programming.");` and `System.out.println("Its easy and simple.");` are used to display information on the standard output device, i.e., the `println() which` is a method of the `out` object, a static member of the class `System`. These statements will display the following strings on the monitor:

```
Welcome to Java Programming.
Its easy and simple.
```

The statements will be printed in separate lines as the method `println()` appends a newline character at the end of the string. However, if we use the `print()` method instead of the `println(), the` newline character is not appended at the end of the string. Also, like C++, every statement in Java must end with a semicolon.

6.4 TYPE CONVERSION

Converting one type of data into another type must follow the rules of *casting*. If a conversion results in the loss of precision, as in an `int` value converted to a `short`, then the compiler will issue an error message unless an explicit cast is made.

The process of converting a value to a wider or higher precision integer or floating point type is called 'numeric promotion'. The Java Virtual Machine (JVM) specification states the following rules for promotion in an expression of two operands, as in `x+i`:

- If either operand is of type `double`, the other is converted to `double`.
- Otherwise, if either operand is of type `float`, the other is converted to `float`.
- Otherwise, if either operand is of type `long`, the other is converted to `long`.
- Otherwise, both operands are converted to type `int`.

Check Your Progress

1. Explain the terms operators and expressions.
2. What are relational operators?
3. Define conditional operators.
4. Explain about the order of precedence of operators.
5. Define about the associativity of operators.

6.5 CONTROL STATEMENTS

A statement is an instruction given to the computer to perform a specific action. In Java, a statement can either be a single statement or a compound statement. A **single statement** specifies a single action and is always terminated by a semicolon ';'. A **compound statement**, also known as a *block*, is a set of statements that are always enclosed within curly braces '{}'.

The control statements are used to control the flow of execution of the program. This execution order depends on the supplied data values and the conditional logic.

By default, statements are executed in the same order in which they appear in the program and each statement is executed only once. However, the serial execution of statements makes a program inflexible and unsuitable for most practical applications. To make a program more flexible, control statements are used to alter the flow of control of the program.

6.5.1 Selection Statements

Conditional statements, also known as **selection** statements, are used to make decisions based on a given condition. If the condition evaluates to `true`, a set of statements is executed, otherwise another set of statement is executed.

The `if` Statement

The `if` statement selects and executes the statement(s) based on a given condition.

The syntax of the `if` statement is as follows:

For a Single Statement

```
if(condition)
statement1;
nextstatement;
```

For a Set of Statements

```
if(condition)
{
statement1;
statement2;
}
nextstatement;
```

Here, if the condition evaluates to `true`, then a given set of statement(s) is executed. However, if the condition evaluates to `false`, then the given set of statements is skipped and the program control passes to the statement following the `if` statement. Figure 6.1 illustrates the flow of control in `if` statement.

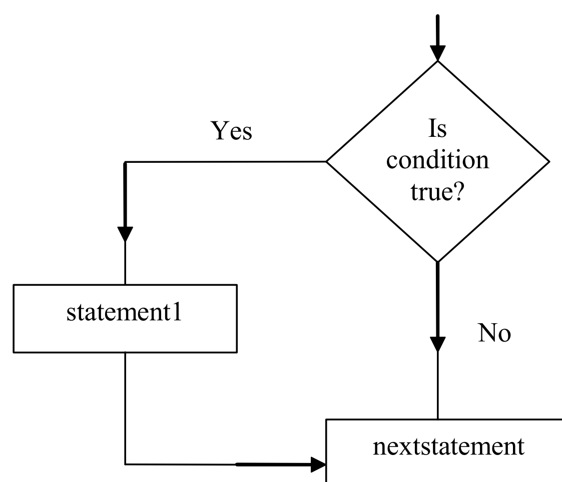


Fig. 6.1 *Flow of Control in if Statement*

The `if-else` Statement

The `if-else` statement causes one of the two possible statement(s) to execute depending upon the result of condition.

The syntax of the `if-else` statements is as follows:

```
if(condition) //if part
{
statement1;
}
else       //else part
statement2;
nextstatement;
```

Here, the `if-else` statement comprises two parts, namely, `if` and `else`. If the condition is `true`, the statements within `if` is executed. However, if the condition is `false`, the statements within `else` is executed as displayed in Figure 6.2.

NOTES

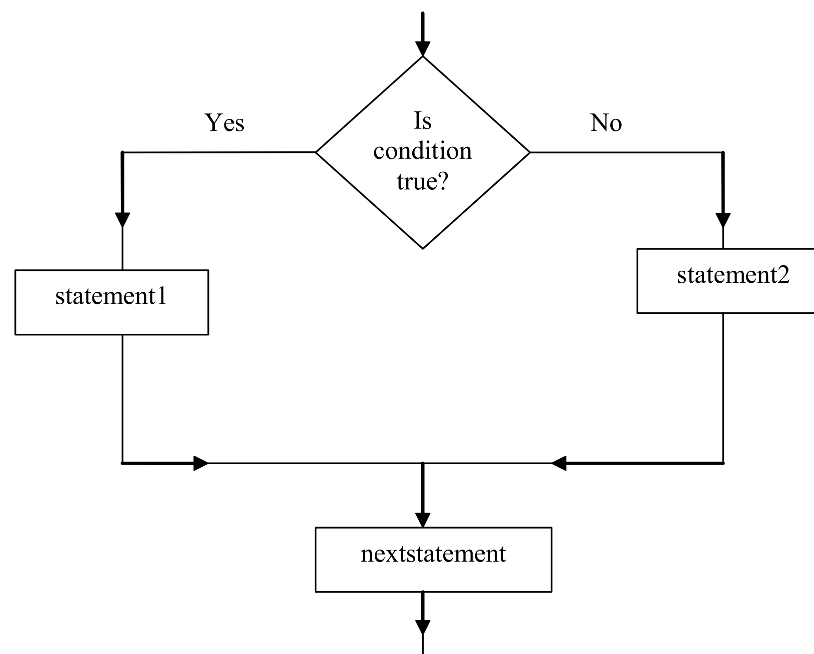


***Fig. 6.2** Flow of Control in `if-else` Statement*

Program 6: A program to demonstrate the use of `if-else` statement.

```
class ConditionalStatement
{
  public static void main (String args[])
  {
     int i=5;
     if (i > 0)
  System.out.println("i is a positive number");
  else
  System.out.println("i is a negative number");
     }
}
```

Output of the program:

```
i is a positive number
```

Nested `if-else` Statement

A nested `if-else` statement contains one or more `if-else` statements. In other words, an `if-else` statement within another `if-else` statement is called a nested `if-else` statement. The `if-else` statement can be nested in three different ways which are as follows:

- The `if-else` statement can be nested within the `if` part.

  The syntax is as follows:

  ```
  if(condition1)
  ```

NOTES

```
{
statement1;
    if(condition2)
       statement2;
else
    statement3;
}
else
statement4;
 nextstatement;
```

- The `if-else` statement can be nested within the `else` part.

The syntax is as follows:

```
if (condition1)
    statement1;
else
{
    statement2;
if (condition2)
       statement3;
else
       statement4;
}
nextstatement;
```

- The `if-else` statement can be nested within both the `if` and the `else` parts.

The syntax is as follows:

```
if(condition1)
{
    statement1;
    if(condition2)
       statement2;
else
    statement3;
}
else
{
    statement4;
if(condition3)
       statement5;
```

```
else
       statement6;
}
nextstatement;
```

NOTES

The `if-else-if` Ladder

The `if-else-if` ladder, also known as the `if-else-if` *staircase*, has an `if-else` statement within the outermost `else` statement. The inner `else` statement can further have other `if-else` statements.

The syntax of the `if-else-if` ladders is displayed in Figure 6.3.

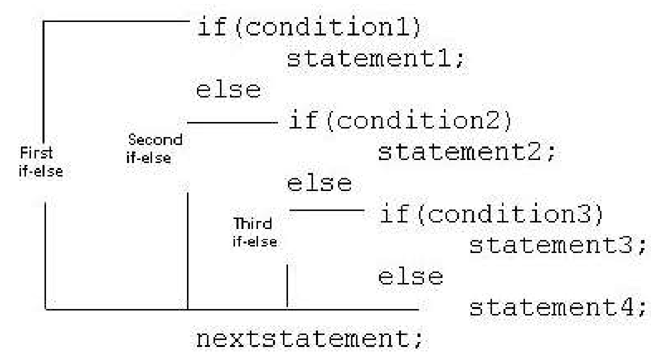
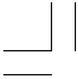
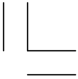


***Fig. 6.3** Syntax of the `if-else-if` Ladder*

Program 7: A program to demonstrate the `if-else-if` statement.

```
import java.io.*;
public class Grade
{
    public static void main(String[] args)
    {
       int marks=0;
       DataInputStream cin=new DataInputStream(System.in);
       try
       {
          System.out.print("Enter the marks: ");
          //reading integer from keyboard
          marks=Integer.parseInt(cin.readLine());
          if(marks>90)
                System.out.println("Grade is A");
          else
if(marks>75)
                System.out.println("Grade is B");
             else
```


```
                if(marks>60)
                 System.out.println("Grade is C");
                else
                 System.out.println("Grade is D");
            }
            catch(Exception e)
            {}
        }
    }
```

Output of the program:

```
Enter the marks: 78
Grade is B
```

Conditional Operator as an Alternative

The conditional operator '`? :`' selects one of the two values or expressions based on a given condition. Due to this decision-making nature of the conditional operator, it is sometimes used as an alternative to the `if-else` statements. Note that the conditional operator selects one of the two values or expressions and not the statements as in the case of an `if-else` statement. In addition, it cannot select more than one value at a time, whereas the `if-else` statement can select and execute more than one statement at a time. For instance, consider the following statement:

```
max=(x>y ? x : y)
```

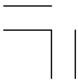
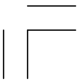
This statement assigns maximum of `x` and `y` to `max`.

The `switch` Statement

The `switch` statement selects a set of statements from the available sets of statements. The `switch` statement evaluates the value of an expression and compares it with the list of integer, character, short or byte constants. It should be noted that the case constants must be compatible with the expression type. When a match is found, all the statements associated with that constant are executed as displayed in Figure 6.4.

The syntax of the `switch` statement is as follows:

```
switch(expression)
{
case <constant1>: statement1;
[break;]
case <constant2>: statement2;
[break;]
case <constant3>: statement3;
[break;]
```

```
[default: statement4;]
}
nextstatement;
```

NOTES

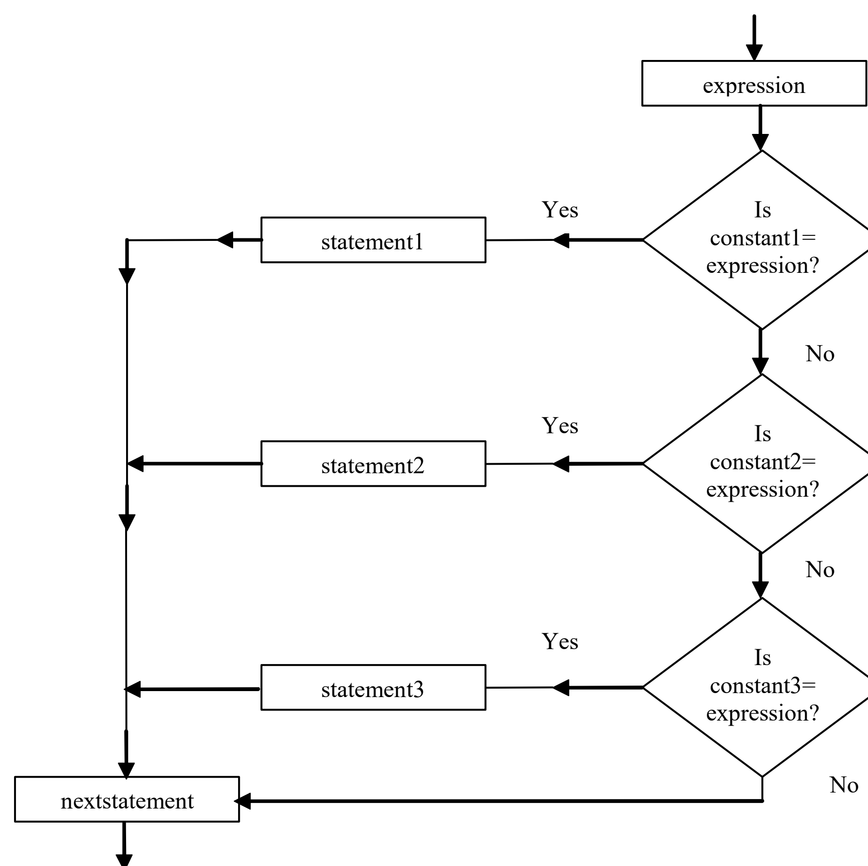


Fig. 6.4 *Flow of Control in* `switch` *Statement*

The Java keywords `case` and `default` provide a list of alternatives. It should be noted that it is not necessary for every `case` label to specify a unique set of statements. The same set of statements can be shared by multiple `case` labels.

The keyword `default` specifies the set of statements to be executed in case no match is found. It should be noted that there can be multiple `case` labels but there can be only one `default` label. However, `default` is an optional statement.

The `break` statements in the `switch` block are optional. However, it is used in the `switch` block to prevent a fall through. **Fall through** is a situation that causes the execution of the remaining cases even after a match has been found. In order to prevent this, `break` statements are used at the end of statements specified by each `case` and `default`. This causes the control to immediately break out of the `switch` block and execute the next statement.

Similar to `if` and `if-else` statements, `switch` statements can also be nested within one another. A nested `switch` statement contains one or more `switch` statements within its `case` label or `default` label (if any).

Note: Switch statement cannot be used for testing floating-point values or string values.

Program 8: A program to demonstrate the use of `switch` statement

```
class SwitchStatement
{
    public static void main(String args[])
    {
       int x=2;
       switch(x)
       {
case 1: System.out.println("Day is Monday");
       break;
case 2: System.out.println("Day is Tuesday");
       break;
case 3: System.out.println("Day is Wednesday");
break;
case 4: System.out.println("Day is Thursday");
break;
case 5: System.out.println("Day is Friday");
break;
case 6: System.out.println("Day is Saturday");
break;
case 7: System.out.println("Day is Sunday");
       break;
default: System.out.println("Invalid option!");
       }
    }
}
```

Output of the program:

```
Day is Tuesday
```

In this illustration, since the value of `x` is `2,` therefore the message `Day is Tuesday` is displayed. In case the value of `x` is 8, the output `Invalid option!` is displayed.

6.5.2 Iteration Statements

The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are known as **iteration**

NOTES

statements. This implies that as long as the condition evaluates to `true`, the set of statement(s) is executed. The various iteration statements used in Java are *for loop*, *while loop* and *do-while loop*.

The `for` Loop

The `for` loop (see Figure 6.5) is one of the most widely used loops in Java. The `for` loop is a deterministic loop, i.e., the number of times the body of the loop is executed is known in advance.

The syntax of the `for` loop is as follows:

```
for(initialize; condition; update)
{
   //body of the for loop
}
```

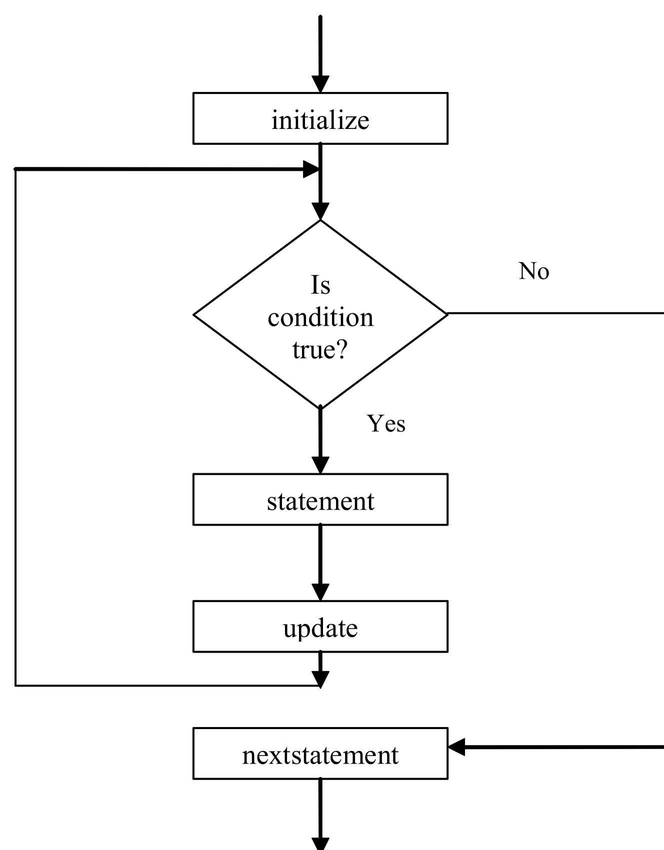
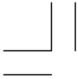
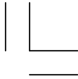


Fig. 6.5 *Flow of Control in* `for` *Loop*

The initialize expression in `for` loop can initialize one or more control variables. A `for` loop can also update more than one variable in its update expression. Note that `initialize`, `condition` and `update` are optional expressions and are always specified in parentheses. All the three expressions are separated

by semicolons. We can also create an infinite loop by excluding all the three expressions as follows:

```
for( ; ; )
{
.
.
}
```

Program 9: A program to display a countdown using `for` loop.

```
class UpdateStatement
{
public static void main(String args[])
{
for (int i=10;i>=1;i—)
{
System.out.print(" "+i);
}
System.out.print("This is an example of for loop");
    }
}
```

Output of the program:

```
10 9 8 7 6 5 4 3 2 1
This is an illustration of for loop
```

`for` Loop using Comma Operator

`for` loop allows multiple variables to control the loop using comma operator. This implies that two or more variables can be used in the `initialize` and the `update` part of the loop. For instance, consider the following statement:

```
for (i=1,j=50;i<10;i++,j—)
```

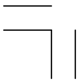
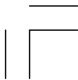
This statement initializes two variables, namely `i` and `j` and updates them. It should be noted that `for` loop cannot have more than one condition separated by a comma.

The `while` Loop

The `while` loop (see Figure 6.6) is used to perform looping operations when the number of iterations is not known in advance. Thus, unlike `for` loop, the `while` loop is non-deterministic in nature.

The syntax of the `while` loop is as follows:

```
while(condition)
{
    // body of the while loop
}
```

NOTES

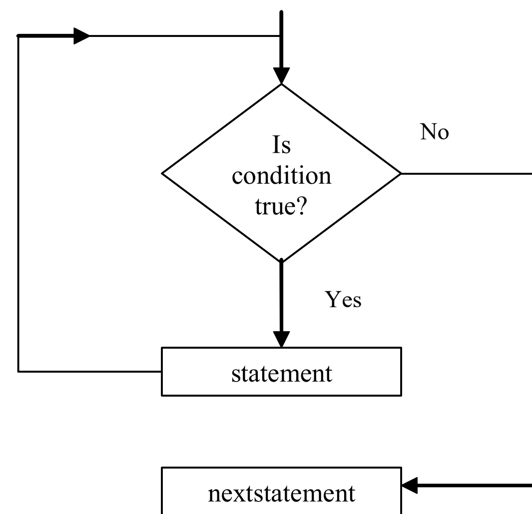


Fig. 6.6 *Flow of Control in* `while` *Loop*

The following are certain important features of the `while` loop:

- Unlike `for` loops where explicit `initialize` and `update` expressions are specified, `while` loops do not specify any explicit `initialize` and `update` expressions. This implies that the control variable must be declared and initialized before the `while` loop and needs to be updated within the body of the `while` loop.
- The `while` loop executes as long as the `condition` evaluates to `true`. If the `condition` evaluates to `false`, then the body of the `while` loop does not execute.

Program 10: A program to determine the sum of first `n` consecutive positive integers.

```
import java.io.*;
public class Sum
{
public static void main(String[] args)
    {
       int n;
       int sum=0;
DataInputStream cin=new DataInputStream (System.in);
       try
       {
          System.out.print("Enter n: ");
//reading input from the user
          n=Integer.parseInt(cin.readLine());
          //loop to calculate the sum
while(n>0)
          {
             sum=sum+n;
             n=n-1;
```

```
            }
System.out.print("The sum is " + sum);
        }
        catch(Exception e)
        {}
    }
}
```

Output of the program:

```
Enter n: 7
The sum is 28
```

The `do-while` Loop

In a `while` loop, the condition is evaluated at the beginning of the loop and if the condition evaluates to `false`, the body of the loop is not executed even once. However, if the body of the loop is to be executed at least once, no matter whether the initial state of the condition is `true` or `false`, the `do-while` loop is used. This loop places the condition to be evaluated at the end of the loop (see Figure 6.7).

The syntax of the `do-while` loop is

```
do
{
//body of do-while loop
}
while(condition);
```

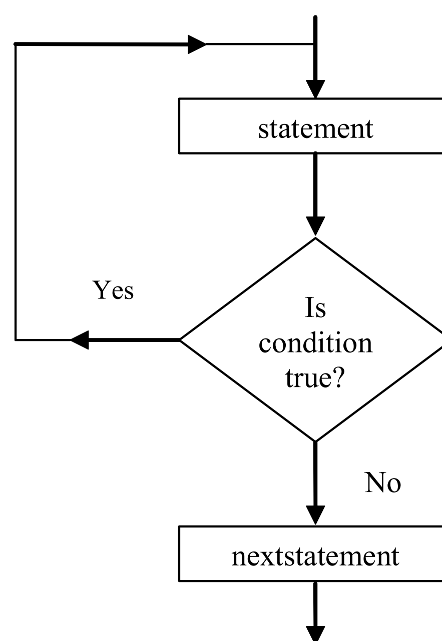


***Fig. 6.7** Flow of Control in `do-while` Loop*

Program 11: A program to calculate the sum of an Arithmetic Progression (AP).

```
class APSeries
{
    public static void main(String args[])
    {
        int first_term=1;
        int number_of_terms=5;
        int term=0;
        int i=1;
        int common_difference=2;
        int sum=0;
        System.out.print("The terms are: ");
        do
        {
            term =first_term+(i-1)*common_difference;
            sum+=term;
            System.out.print(" "+term);
            ++i;
        }
        while(i<=number_of_terms);
        System.out.println();
        System.out.println("The sum of A.P is: " +sum);
    }
}
```

Output of the program:

```
The terms are: 1 3 5 7 9
The sum of A.P is: 25
```

Nested Loops

Nested loops are the loops present within the body of another loop. All the three loops (`for`, `while` and `do-while`) can be nested.

Program 12: A program to demonstrate the nested `for` loop.

```
class NestedLoop
{
    public static void main(String args[])
    {
        int a,b;
        for(a=0;a<5;a++)              //outer loop
        {
            for(b=a;b<5;b++)    //inner loop
            {
                System.out.print("*");
            }
                System.out.println();
        }
    }
}
```

Output of the program:

```
*****
****
***
**
*
```

6.5.3 Jump Statements

Jump statements are used to alter the flow of control unconditionally. Thus, jump statements transfer the program control unconditionally. The jump statements defined in Java include `break`, `continue` and `return`.

The `break` Statement

The `break` statement is extensively used in loops and `switch` statements. It immediately terminates the loop or the `switch` statement, bypassing the remaining statements. The control then passes to the statement that immediately follows the loop or the `switch` statement. A `break` statement can be used in any of the three Java loops. In case of nested loops, the break will exit the only the single loop, i.e., the loop in which it is placed.

The `continue` Statement

The `continue` statement is used to 'continue' the loop with its next iteration. In other words, it skips any remaining statements in the current iteration and immediately passes the control to the next iteration. It does not terminate the loop (as in the case of `break` statements); rather it only terminates the current iteration of the loop. Like a `break` statement, a `continue` statement can be used in any of the three loops.

The `return` Statement

The `return` statement is used to transfer the control out of the method explicitly. It transfers the control back to the caller and terminates the method in which it is present. When the `return` statement is encountered in the `main()` method, it transfers the control back to the Java run-time system and terminates the program execution.

Program 13: A program to add the factors of a number using `break` and `continue` statements.

```
class Jump
{
    public static void main(String args[])
    {
        int factor=0, number=10, sum=0;
        System.out.println("Number=" +number);
        while(true)
```

NOTES

```
        {
            factor++;
            if(factor>number)
                break;
            if(number%factor!=0)
                continue;
            sum=sum+factor;
        }
        System.out.println("Sum of factors=" +sum);
        if(sum>0)
            return;
        System.out.println("This statement is not
executed");
    }
}
```

Output of the program:

```
Number=10
Sum of factors=18
```

6.6 LABELED LOOPS

`Break` and `continue` statements allow the programmer to break out of the loop. However, they do not allow you to simply jump to another part of the program or out of the nested loop or `switch` statement. Java allows the programmer to jump from one block of statement to another with the help of labels. A label is an identifier which must follow the rules for naming identifiers in Java. It can be placed before the block of statements or loop followed by a colon (:). For instance, consider the following statement:

```
LabelName: for( ; ; )
        {
            .
            .
    }
```

The `break` statement passes the control out of the innermost loop or the innermost `switch` statement and the `continue` statement continues with the next iteration of the innermost loop only. However, with the help of labels, `break` statement can be used to cause the control to jump out of the outer loop or `switch` statement. For instance, consider the following statements.

```
outerloop:      for(i=0;i<5;i++)
            {
innerloop:          for(j=1;j<5;j++)
                {
                    System.out.println("*");
                    if(i=j)
                    break outerloop;
```

```
                    .
                    .
                }
            }
```

NOTES

In this case, the `break` statement placed inside the `innerloop` will force the control to jump out of both the inner and outer loops. Similarly, with the help of labels, the `continue` statement can be used to continue with the next iteration of the outer loop.

Program 14: A program to demonstrate the use of `continue` statement in a labeled loop.

```
class LabelExample
{
    public static void main(String args[])
    {
        outerloop: for(int i=0;i<3;i++)
              {
                   for(int j=0;j<5;j++)
                   {
                       if(j>i)
                           continue outerloop;
                             System.out.println(" i
                             "+i+" j "+j);
                   }
              }
              System.out.println("Loop Ends");
    }
}
```

Output of the program:

```
i 0 j 0
i 1 j 0
i 1 j 1
i 2 j 0
i 2 j 1
i 2 j 2
Loop Ends
```

In this program, if we substitute `continue` statement with the `break` statement, the following output will be generated.

```
i 0 j 0
Loop Ends
```

Check Your Progress

6. What are control statements?
7. Explain conditional operator as an alternative.
8. Define iteration statements.
9. What is a label?

6.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Operators are the symbols which perform operations on various data items known as operands. For instance, in `a + b`, `a` and `b` are operands and + is an operator. Note that to perform an operation, operators and operands are combined together forming an ***expression***. For instance, to perform an addition operation on operands `a` and `b`, the addition (+) operator is combined with the operands `a` and `b` forming an expression.

   Depending on the function performed, the Java operators can be classified into various categories. These include arithmetic operators, increment and decrement operators, relational operators, logical operators, conditional operators, assignment operators, bitwise operators and special operators.

2. Relational operators are used for comparing two values or expressions. The various relational operators provided by Java include less than ‘<’, less than or equal to ‘<=’, greater than ‘>’, greater than or equal to ‘>=’, equal to ‘==’ and not equal to ‘ != ’ operator. They return values of boolean type, i.e., either `true` or `false`. For instance, consider two variables `a` and `b` having values 20 and 30 respectively. In this case, the expression `a<b` returns `true` whereas the expression `a>b` returns `false`.
3. The conditional operator selects a value based on a specified condition. Note that the conditional operator is a ternary operator, i.e., this operator involves three operands.

   The syntax of the conditional operator is as follows:

   ```
   expression1 ? expression2: expression3
   ```

4. The order or priority in which various operators in an expression are evaluated is known as **precedence**. Every operator in Java has a precedence associated with it. The operators with a higher precedence are evaluated before the operators with a lower precedence. For instance, multiplication is performed before addition as the multiplication operator has higher precedence than the addition operator.
5. The order or priority in which operators of the same precedence are evaluated is known as **associativity**. For instance, the addition and subtraction operators have the same precedence. However, addition or subtraction may be performed on the expression depending upon the order of their occurrence.
6. A statement is an instruction given to the computer to perform a specific action. In Java, a statement can either be a single statement or a compound statement. A **single statement** specifies a single action and is always

terminated by a semicolon ';'. A **compound statement**, also known as a *block*, is a set of state ments that are always enclosed within curly braces '{}'.

7. The conditional operator '`? :`' selects one of the two values or expressions based on a given condition. Due to this decision-making nature of the conditional operator, it is sometimes used as an alternative to the `if-else` statements. Note that the conditional operator selects one of the two values or expressions and not the statements as in the case of an `if-else` statement. In addition, it cannot select more than one value at a time, whereas the `if-else` statement can select and execute more than one statement at a time.
8. The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are known as **iteration statements**. This implies that as long as the condition evaluates to `true`, the set of statement(s) is executed. The various iteration statements used in Java are *for loop*, *while loop* and *do-while loop*.
9. A label is an identifier which must follow the rules for naming identifiers in Java. It can be placed before the block of statements or loop followed by a colon (**:**).

6.8 SUMMARY

- Java operators can be classified into various categories. These include arithmetic operators, increment and decrement operators, relational operators, logical operators, conditional operator, assignment operators, bitwise operators and special operators.
- Control statements are used to alter the flow of control of the program. In Java, the control statements are broadly classified into three categories, namely, conditional statements, iteration statements and jump statements.
- The `for` loop is one of the most widely used loops in Java. The `for` loop is a deterministic loop, that is, the number of times the body of the loop is executed is known in advance. The `while` loop is used to perform looping operations when the number of iterations is not known in advance. In a `while` loop, the condition is evaluated at the beginning of the loop and if the condition evaluates to `false`, the body of the loop is not executed even once. The `do-while` loop is used if the body of the loop is to be executed at least once, no matter whether the initial state of the condition is `true` or `false`.
- Nested loops are the loops present within the body of another loop.

- Jump statements are used to alter the flow of control unconditionally; that is, jump statements transfer the program control unconditionally. The jump statements defined in Java are `break`, `continue` and `return`.
- Labels are used to jump from one block of statement to another.

6.9 KEY WORDS

- **Statement:** An instruction given to the computer to perform a specific action.
- **Single statement:** A statement that specifies a single action and is always terminated by a semicolon ';'.
- **Compound statement:** A set of statements that are always enclosed within curly braces '{}'.
- **Conditional statements:** Statements that are used to make decisions based on a given condition.
- **Iteration statements:** Statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied.
- **Nested loops:** Loops present within the body of another loop.
- **Jump statements:** Loops used to alter the flow of control unconditionally.
- **Precedence:** The order or priority in which various operators in an expression are evaluated.
- **Associativity:** The order or priority in which operators of the same precedence are evaluated.

6.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. State the difference between the execution of `while` and `do-while` loop. Support your answer with suitable example.
2. Define the significance of the type conversion.
3. What are the use of arithmetic operators in Java programs?
4. Differentiate between the simple assignment operator and compound operator.
5. Compare between `instanceof` and dot operator.

NOTES

Long-Answer Questions

1. Discuss the operator precedence and associativity in Java programs.
2. Elaborate the importance of decision-making and branching statements.
3. What is the difference between precedence and associativity? Evaluate the following expressions.
   (a) `20*3+(-10/2)`
   (b) `15-16/4+6`
   (c) `10*(15+8)/2`
4. Write a program to read two integer numbers from the keyboard and display their sum on the screen.
5. What are control statements? Explain the different types of control statements with examples?
6. Consider the following code segment

```
if(x%2>=0)
{
   if(x%2==0)
      System.out.println("x is an even number");
   if(x%2>0)
      System.out.println("x is an odd number");
}
  Rewrite the code using:
```

   (a) `if-else` statement (b) conditional operator
7. Write a program to find and print the multiples of 9 between 100 and 200 using `for` loop.
8. Explain the use of `break` and `default` in `switch` statements.
9. Write a program to print the following output

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

10. Write a program to check whether the number is palindrome or not. (Hint: Palindrome numbers are those which read the same from left to right and from right to left.)
11. Write a program to calculate the sum of this series.
    $1 + x + x^2 + x^3 + \ldots + x^n$
12. Write a program to multiply two given matrices and print the resultant matrix.

NOTES

6.11 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

BLOCK - III
CLASSES, OBJECTS AND METHODS

NOTES

UNIT 7 CLASSES IN JAVA

Structure

7.0 INTRODUCTION

In this unit you will be familiarized with classes and objects. The key objective of Object Oriented Programming (OOP) is to represent the various real world objects as program elements. In Java, this objective is accomplished with the help of a class that binds the data and the methods to manipulate that data together under a single entity. All the OOP concepts, such as data abstraction, encapsulation, inheritance and polymorphism are implemented with the help of classes. A class serves as a template that provides a layout common to all of its instances known as objects. That is, a class is only a logical abstraction that specifies what data and methods its objects will have whereas the objects are the physical entities through

NOTES

which those data and methods can be used in a program. Thus, objects are considered as the building blocks of object oriented programming. This unit discusses the concept of classes and objects in detail. Constructors are the special methods that are used to initialize objects at the time of their creation. The Java programming language allows you to define a class within another class. Such a class is called a nested class. A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private. Static nested classes do not have access to other members of the enclosing class.

In this unit, you will study about the classes in Java, defining a class, methods, creating objects, accessing class members, constructors, methods overloading, static members, nesting of methods, inheritance, overriding methods, final variables, classes and methods.

7.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain classes and objects that form the basis of object oriented programming
- Understand the process of accessing the members of a class
- Learn about the concept of argument passing
- Comprehend the uses of constructors
- Know about method overloading
- Discuss about final variables
- Explain the significance of nested classes and inheritance

7.2 DEFINING A CLASS

A class is a user defined data type that can be used to create instances of its type called objects. Like any other user defined data type, it also needs to be declared and defined in the program. A class definition specifies a new data type that can be treated as a built-in data type.

The syntax for defining a class is,

```
class class_name
{
    //variables declaration
    //methods declaration
}
```

The variables declared in the class are known as instance variables. The variables and methods declared within the curly braces are collectively known as members of the class.

NOTES

A class can also be empty. That is, the class definition given below is also valid.

```
class class_name
{
}
```

Here, since the body of the class is empty, it does not contain any variables and methods so it cannot perform any useful action. However, this class can be successfully compiled and we can also create objects using it.

Note: In Java, there is no semicolon after closing brace in class definition.

For example, a simple class definition without any method is shown below:

```
class Cuboid
{
int length;
int width;        //variables declaration
int height;
}
```

In this example, a `class` named `Cuboid` with three instance variables of type `int`, namely, `length`, `width` and `height` is created.

7.2.1 Defining Methods

As you would have already studied, a class consists of instance variables and methods. A class that consists of only variables (and without methods which manipulate them) cannot perform any useful operation. Therefore, to access the instance variables of a class and manipulate them, you must add methods in the class.

The syntax for defining a method is,

```
return_type method_name(parameter_list)
{
body of the method
}
```

where,

`return_type` is the type of data that is returned by the method.

`method_name` specifies the name of the method. This can be any name other than the keywords in Java.

`parameter_list` consists of a series of pairs of data type and identifiers separated by commas.

Note: The `parameter_list` can be empty and if a method does not return any value, its return type must be `void`.

NOTES

For example, consider the following method definition is shown below:

```
int volume()
{
body of the method
}
```

Here, the method `volume()` does not accept any parameter and returns a value of type `int`.

For example, a class definition with method is shown below:

```
class Cuboid
{
int length;
int width;        //variables declaration
int height;
int volume()      //method definition
{
    return(length*width*height);
}
```

Note: Methods must be declared immediately after the declaration of the instance variables inside the body of the class.

7.2.2 Creating Objects

Once a class is defined, it can be used to create variables of its type known as objects. The relationship between an object and a class is the same as that of any variable and its data type. To create an object, we need to declare it first.

The syntax for declaring an object is:

```
class_name object_name;
```

where,

`class_name` is the name of the class.

`object_name` is the name of the object of `class_name` type.

For example, the statement to declare an object of `Cuboid` type is:

```
Cuboid cobj;
```

This statement declares a variable `cobj` as a reference to an object of `Cuboid` type. After the execution of this statement, `cobj` contains the `null` value, which indicates that it does not point to an actual object. Once the object is declared, we need to create it by allocating the required memory space to it. In Java, this is done with the help of the `new` operator.

The syntax for creating an object is:

```
object_name=new class_name;
```

For example, the statement to create an object of `Cuboid` type is:

```
cobj=new Cuboid();
```

This statement creates an actual object by dynamically allocating memory space to it and returns a reference to `cobj`.

The two given statements can be combined into one statement as follows:

```
Cuboid cobj=new Cuboid ();
```

Note: Since an object is an instance of a class, the process of creating an object of a class is known as instantiation.

7.3 ACCESSING MEMBERS OF A CLASS

Each object of a class has its own set of variables. These variables should be assigned values before being used in the program. The instance variables and methods added in the program cannot be accessed directly outside the class using their names. To access the variables and methods outside the class, the dot (`.`) operator is used as follows:

```
object_name.variable_name
object_name.method_name(parameter_list)
```

where,

`object_name` is the name of the object.

`variable_name` is the name of the instance variable that is to be accessed.

`method_name` is the name of the method which is to be called.

`parameter_list` is the series of pairs of data types and their respective identifiers.

For example, the instance variable `length` of `Cuboid` class can be accessed as follows:

```
cobj.length;
```

Similarly, the method `volume()` of `Cuboid` class can be accessed as follows:

```
cobj.volume();
```

Program 1: A program to demonstrate the accessing of members of a class.

```
class Cuboid
{
    int length;
    int width;
    int height;
    int volume()   //method definition
    {
       return(length*width*height);
```

NOTES

```
    }
}
class ClassDemo
{
    public static void main(String args[])
    {
        Cuboid cobj=new Cuboid();    //object creation
        cobj.length=60;
        cobj.width=20;           //accessing variables
        cobj.height=40;
        int vol=cobj.volume();  //calling method
        System.out.println("The volume of the cuboid is: "
        +vol);
    }
}
```

Output of the program:

```
The volume of the cuboid is: 48000
```

In this example, the instance variables `length`, `width` and `height` of the object `cobj` are assigned values outside the class using the dot operator. Alternatively, the instance variables can be assigned values by using a parameterized method.

7.4 CONSTRUCTORS

A constructor is a special method which is used to initialize objects at the time of their creation. The name of the constructor is the same as the name of the class in which it resides. Unlike other methods in Java, a constructor does not have any return type (not even `void`). This is because the implicit return type of constructor is the instance of the class to which it belongs.

The syntax to define a constructor is:

```
class class_name
{
    //Variable declaration
    class_name(parameter_list) //constructor definition
    {
        //body of the constructor
    }
}
```

It should be noted that if you do not explicitly define any constructor, then the Java compiler automatically provides a default constructor that initializes all the instance variables to 0.

Program 2: A program to demonstrate the concept of a constructor.

```
class Cuboid
{
int length;
int width;
int height;
Cuboid()     //constructor definition without parameters
    {
length=20;
width=10;
height=15;
}
    int volume()
    {
       return (length*width*height);
    }
}
class Constructor
{
    public static void main(String args[])
    {
       Cuboid c1=new Cuboid();
       Cuboid c2=new Cuboid();
       int a=c1.volume();
       int b=c2.volume();
System.out.println("The volume of first cuboid is " +a);
System.out.println("The volume of second cuboid is " +b);
    }
}
```

NOTES

Output of the program:

```
The volume of first cuboid is 3000
The volume of second cuboid is 3000
```

In this example, the constructor initializes both the objects `c1` and `c2` of `Cuboid` type with the same values, as a result of which the volume of both the cuboids is the same. Clearly, a non-parameterized constructor provides the same values to all the objects of a class.

Parameterized Constructors

When different objects of a class need to be initialized with different values, a parameterized constructor can be defined. A parameterized constructor is a

NOTES

constructor that accepts one or more parameters at the time of creation of objects and initializes the instance variables of the objects with these parameters. It makes the program more flexible because by using it we can assign different values to the instance variables of the objects of a class.

7.5 METHOD OVERLOADING

In some cases, when a similar action is to be performed on various types of data, different methods having different names needs to be defined for all types of data. This makes the program very complex as the programmer must keep track of the names of all the methods. To prevent such situations, Java allows the methods to be overloaded.

Method overloading allows multiple methods to have the same name with different parameters and definitions. The application of method overloading is found in those programs where objects in a class are needed to perform similar operations but using different input parameters. When a method is called, the Java compiler compares the method name and then the type and number of arguments associated with it, to determine which of the method is to be called and executed. This process is known as polymorphism.

In order to create overloaded methods in your program, you have to create different method definitions in the class, all with the same name but having different parameters. The different methods are identified on the basis of the number of parameters, the data type of the parameters or the order of the data type of the parameters passed to the method.

To understand the concept of method overloading, consider these method declarations.

```
void sum(int);
void sum(int,int);
```

In these statements, the methods are different as the number of arguments passed are different. Now, consider these statements.

```
void addition(int);
void addition(char);
```

In these statements, the two methods are different as the data type of arguments passed is different. Now, consider these statements.

```
void area(int,float);
void area(float,int);
```

In these statements, the two methods are different as the order of the data type of the arguments passed is different.

NOTES

Program 3: A program to demonstrate the concept of method overloading.

```
class Shapes
{
    void perimeter(int a)//method with int type parameter
    {
System.out.println("The perimeter of the square is "+(4*a));
    }
void perimeter(double a)//method with double type parameter
    {
System.out.println("The circumference of the circle is
"+(2*a*(22/7)));
    }
    void perimeter(int a, int b) //method with two parameters
    {
System.out.println("The perimeter of the rectangle is
"+2*(a+b));
    }
}
class MethodOverloading
{
    public static void main(String args[])
    {
       Shapes s=new Shapes();
       s.perimeter(5.5);
       s.perimeter(10);
       s.perimeter(15,10);
    }
}
```

Output of the program:

```
The circumference of the circle is 33.0
The perimeter of the square is 40
The perimeter of the rectangle is 50
```

7.6 STATIC MEMBERS

So far, we have seen that members of a class can be initialized or accessed by its objects only. However, it is possible to define members of a class without reference to a specific object. This is done by declaring the instance variables and methods of a class as `static`. Unlike non-static members, the `static` members are associated with the class as a whole, rather than with individual objects. Therefore, outside

class, its static members are called by using class names rather than by objects. Since the static variables and methods contain the properties of a class and not of the individual objects, they are also known as class variables and class methods respectively.

NOTES

The syntax to declare static variable is:

```
static return_type variable_name;
```

The syntax to access static variable is:

```
class_name.variable_name
```

The syntax to declare static method is:

```
static return_type method_name(parameter_list);
```

The syntax to access static method is:

```
class_name.method_name
```

If a class member is declared as `static`, only one copy of that data member is created, regardless of the number of objects. All the objects of a class share this single copy of the `static` data member.

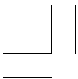
Program 4: A program to demonstrate the use of static members of a class.

```
class StaticMembers
{
    static int i=10;            //static variable
    static void add(int x, int y)   //static method
    {
       System.out.println("Sum of two numbers is: "+(x+y));

    }
}
class StaticExample
{
    public static void main(String args[])
    {
       StaticMembers.add(30,40);    //calling  static
method
       int i= StaticMembers.i; //accessing static
                      //variable
       System.out.println("Value of i is: "+i);
    }
}
```

Output of the program:

```
Sum of two numbers is: 70
Value of i is: 10
```




Here, variable `i` and method `add()` are declared as `static`. Therefore, outside the class they are accessed using the name of the class in which they are defined.

Note: The methods declared as `static` can access only `static` data and call `static` methods.

`Final` Keyword

A variable which has a constant value or a method that cannot be overridden in a subclass or a child class is specified by a `final` keyword. The syntax for a `final` keyword is shown as follows:

```
final public int a  = 10;
final public void classMethod( );
```

This keyword is used to reference the current object inside a class definition by passing the need for an instance variable.

`Static` Keyword

Sometimes you need a common variable or method for all the objects derived from a class. The `static` keyword specifies that a variable or a method is the same for all objects of a particular class.

Each time you create an object from a class, space is allocated for the variables. If a variable is declared `static`, space is allocated only once, i.e., the first time when you create the object. This space is shared by all subsequent objects. `Static` method is one whose mention is the same for all objects. `Static` method has access to `static` variables only. The syntax for declaring a `static` variable or method is as follow:

```
static int a;
static void classMethod( );
```

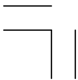
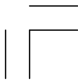
Check Your Progress

1. Define a class in object oriented programming.
2. What are instance variables?
3. Name the two ways by which an argument can be passed to the method.
4. Define a constructor.
5. What is a parameterized constructor?
6. What is the difference between static and non-static member of a class?

7.7 NESTED CLASSES

The Java programming language allows you to define a class within another class. Such a class is called a nested class as shown below:

```
class OuterClass {
```

NOTES

```
    ...
    class NestedClass {
        ...
    }
}
```

Nested classes are divided into two categories: static and non-static. Nested classes that are declared `static` are simply called `static` nested classes. Non-static nested classes are called inner classes.

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private. Static nested classes do not have access to other members of the enclosing class. As a member of the `OuterClass`, a nested class can be declared `private`, `public`, `protected` or package private.

The following are some of the reasons of using nested classes:

- **Logical Grouping of Classes:** If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such 'helper classes' makes the package more streamlined.
- **Increased Encapsulation:** Consider two top level classes, A and B, where B needs access to members of A that would otherwise be declared `private`. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.
- **More Readable, Maintainable Code:** Nesting small classes within top level classes places the code closer to where it is used.

`Static` Nested Classes

A `static` nested class is associated with its outer class as with class methods and variables. A `static` nested class cannot refer directly to instance variables or methods defined in its enclosing class like `static` class methods it can use them only through an object reference. A `static` nested class interacts with the instance members of its outer class (and other classes) just like any other top level

class. In effect, a `static` nested class is behaviorally a top level class that has been nested in another top level class for packaging convenience.

`Static` nested classes are accessed using the enclosing class name:

```
OuterClass.StaticNestedClass
```

For example, to create an object for the `static` nested class, use this syntax:

```
OuterClass.StaticNestedClass nestedObject = new
OuterClass.StaticNestedClass ();
```

Non-Static / Inner Classes

An inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. An inner class cannot define any static members itself because it is associated with an instance. Objects that are instances of an inner class exist within an instance of the outer class. Consider the following classes:

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
}
```

An instance of `InnerClass` can exist only within an instance of `OuterClass` and has direct access to the methods and fields of its enclosing instance. Figure 7.1 illustrates this idea.

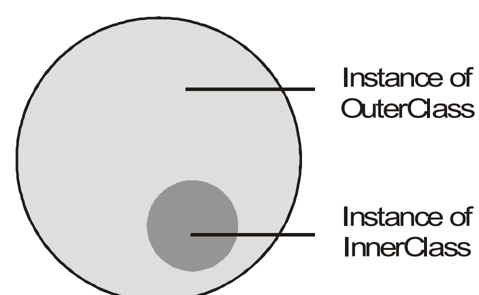


Fig. 7.1 *An Instance of* `InnerClass` *Exists Within an Instance of* `OuterClass`

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

```
OuterClass.InnerClass innerObject = outerObject.new
InnerClass();
```

The following are the categories of inner classes:

- **Member Class:** They are declared outside a function, hence considered as a 'member' and not declared '`static`'.

NOTES

- **Local Class:** These are classes that are declared in the body of a function. They can only be referred in the rest of the function. They can use local variables and parameters of the function, but only ones that are declared '`final`'. This is because the local class instance must maintain a separate copy of the variable, so as not to have the ambiguity of two modifiable variables with the same name in the same scope, the variable is forced to be non-modifiable.
- **Anonymous Class:** These are local classes that are automatically declared and instantiated in the middle of an expression. They can only directly extend one class or implement one interface. They can specify arguments to the constructor of the superclass, but cannot otherwise have a constructor.

Program 5: A Java program shows how to define and use an inner class:

```
class Outer
{
    int outer_x = 10;
   void test( )
   {
    Inner inner = new Inner( );
    inner.display( );
    }
class Inner
{
    void display( )
    {
    System.out.println("Display : outer_x  =" + outer_x);
    }
    }
}
class InnerClass
{
    public static void main(String args [ ])
    {
    Outer outer =  new Outer( );
    outer.test( );
    }
}
```

Output of the program:

```
Display : outer_x  = 10
```

7.8 INHERITANCE

NOTES

Inheritance is an Object Oriented Program (OOP) term in which the non-private features and attributes of a given superclass are made available to its subclass (es).

How is Inheritance Useful?

Through Inheritance code duplication can be avoided by reusing the code (attributes and methods) of one class in another class. You can derive new classes from already existing classes.

Program 6: A Java program shows the usage of inheritance:

```
class SuperBazar
{
    int sales;
    public SuperBazar( )  {  }   / / default constructor
    public SuperBazar(int bill)
    {
       sales = bill;
    }
    public void tax( )
{
    double tax_to_pay = sales * 8.33 /100;
    System.outprintln("Customer sales amount is Rs.   "
+ sales);
    System.outprintln("Tax to pay is Rs.  " + tax_to_pay);
       System.outprintln("Total Bill amount  is Rs.  " +
(sales + tax_to_pay));

       }
}

public class SalesTax extends SuperBazar
{
    public SalesTax(int amount)
    {
       sales = amount;
    }
public static void main(String args[ ])
{
    SalesTax apgst = new SalesTax(5000);
    apgst.tax( );
```

```
}
}
```

Output of the program:

NOTES

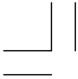
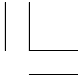
```
Customer Sales amount is Rs. 5000
Tax to pay is Rs. 416.5
Total Bill amount is Rs. 5416.5
```

7.8.1 Super Keyword

`Super` is a Java keyword used to reference non-static methods or variables of the parent class that may be hidden by the current class. This keyword is a method (i.e., `Super()`).

Program 7: A Java program using `super` Keyword:

```
class Number1
{
    int num = 10;
    public void display( )
    {
       System.out.println("From Number 1 : num =  " +
num);
    }
}
public class Number2 extends Number1
{
Super (num);
    int num = 20;
    public void show( )
    {
       System.out.println("From Number 2 without super
num =   "  + num);
       System.out.println("From Number 2 with super num =
"  + num);
    }
    public static void main(String args[ ])
    {
       Number1 number1 = new Number1( );
       Number2 number2 = new Number2( );
       number1.display( ); / / superclass object calling
its own class method
       number2.display( ); /  /subclass  object  calling
superclass method
       number2.display( ); / / subclass object calling its
own class method
```


```
    }
}
```

Output of the program:

```
From Number  1  :  num = 10
From Number  2 without super num = 20
From Number  2 with super num = 10
```

NOTES

In this program, both superclass and subclass contain the variables by the same name **num**. Super keyword when used in a subclass hides the variable num of subclass and access the variable num of superclass

Note: Super( ) should be the first statement, if used in a subclass method. Otherwise compiler indicates an error.

7.8.2 Multilevel Hierarchy

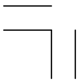
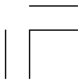
You have been using simple class hierarchies that consist of only a superclass and a subclass. However, you can build hierarchies that contain many layers of inheritance. To see how a multilevel hierarchy can be useful, consider the following program.

Program 8: A Java program shows the use of multilevel hierarchy.

```
class Worker
{
    int salary;
    float overtime;
    double total;

    Worker( )     { }

    publicWorker(int x, float y)
    {
       salary = x;
       overtime = y;
    }
    public double totalSalary( )
    {
       return total = salary + overtime;
    }
    public double netSalary( )
    {
       return totalSalary( ) - totalSalary( ) * 30 / 100;
    }
}
```

NOTES

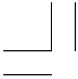
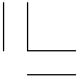
```
public class Supervisor extends Worker
int  supervisory_allowance;
public Supervisor(int x, float y, int z)
{
salary=x;
overtime=y;
supervisory_allowance = z;
}
public static void  main (String args[ ] )
{
Worker shopfloor =new worker (2500,376.5f);
Worker  frontoffice=new worker(2200,85.00f);
Supervisor gmoffice=new supervisor(5700,459.25f.1876);
Supervisor planningdept=newsupervisor(6780.145.85f,2567);
double d;
d=shopfloor.totalSalary( );
System.out.println("Total Salary of shopfloor worker is
Rs." +d);
System.out println("Total Salary of frontoffice worker
is Rs"+ frontoffice.totalSalary(  ));
d=gmoffice.total Salary( );
System.out println ("Total  Salary of  GM office supervisor
is Rs" + d);
System.out.println("Total Salary of Planning department
supervisor is Rs." + planning dept.totalSalary( ));
System. out. println("Supervisory allowance of gmoffice
is Rs." +gmoffice. supervisory _  allowance );
System.out.println ("Supervisory allowance of  planningdept
is Rs." + planningdept.supervisory_allowance);
}
```

Output of the program:

```
Total Salary of shopfloor worker is Rs. 2876.5
Total Salary of frontoffice worker is Rs. 2285.0
Total Salary of GM office supervisor is Rs. 6159.25
Total Salary of Planning department supervisor is Rs.
6925.85009765625
Supervisor allowance of gmoffice is Rs. 1876
Supervisor allowance of planningdept is Rs. 2567
```

7.8.3 Method Overriding

When a superclass and subclass have the same method signature (including return type) for a method, then the method in the subclass is said to override the method

NOTES

of superclass. This involves changing the method body in the subclass. Sometimes it is useful to override a parent method in a derived class.

For example:

```
class Area1
{
    public int area(int x)
    {
       return x * x;
    }
}
class Area2 extends Area1
{
    public int area(int x)
    {
    return x + x;
    }
}
```

In this example, `method area (int x)` is same in both the subclass and superclass, however they serve two different functions (in one addition and in the other multiplication). The method area is said to be overridden in subclass. When an object of the subclass calls the area method, the method of subclass is called hiding the superclass method.

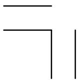
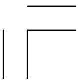
How is the Method of Overriding Useful in OOPs?

Overriding is useful to avoid new method names when the methods perform closely related functions under different conditions. With overriding you can provide different functionalities for the same methods. This leads to polymorphism and increases the program readability in big programs.

Program 9: A Java program for overriding

```
class  area
  {
      public void area(int r)
      {
      System.out.println(" Area of circle:  "  + Math.round
  ( Math.PI * r *r));
      }
      public void area(int l, int b)
      {

  System.out.println(" Area of rectangle:   "  + 2 * (l +
  b));
```

NOTES

```
    }
}
public class Perimeter extends Area
{
    public void area(int r)
    {
      System.out.println(" Perimeter of circle:   "  +
Math.round ( 2 * Math.PI * r ));
    }
public static void main(String args[ ] )
{
    Area a = newArea( );
    Periemter p = new Perimeter ( );
    a.area (22);
    p.area (22);
    p.area (22, 33);
}
}
```

Output of the program:

```
Area of circle:  1521
Perimeter of circle:  138
Area of rectangle:  110
```

7.8.4 Dynamic Method Dispatch

Dynamic method dispatch is the mechanism by which a call to an overridden method is determined at runtime instead at the time of compiling the program. This is how Java implements runtime polymorphism.

Program 10: A Java program uses dynamic method dispatch.

```
class Animal
{
    public void speak( )
    {
      System.out.println(" See how different animals cry
with Polymorphism");
    }
}
class Cat extends Animal
{
    public void speak( )
    {
```

```
      System.out.println("The Cat speaks: meow meow");
    }
}
class Dog extends Cat
{
    public void speak( )
    {
      System.out.println("The Dog speaks: bow bow");
    }
}

class Cow extends Dogl
{
    public void speak( )
    {
      System.out.println("The Cow speaks: amba amba");
    }
}
public class AnimalCry
{
    public static void main(Sting args[ ] )
    {
Animal animal; / / creating a superclass reference variable,
animal
/ / and observe that new is not used here in creating / /
a reference variable
Cat cat = new Cat( ); / / Cat object created animal = cat;
                 / / Cat object is assigned to animal
animal.speak( ); / / Now animal calls the method speak of
cat
Dog dog = new Dog( );/ /  Dog object created animal = dog;
                 / /  Dog object is assigned to animal
animal.speak( ); / /  Now animal calls the method speak
of dog
Cow cow = new Cow( );/ / Cow object created
animal = cow;    / / Cow object is assigned to animal
animal.speak( ); / /  Now animal calls the method speak
of cow
    System.out.println(" From main : Cries are over");
    }
}
```

NOTES

NOTES

Output of the program:

```
See how different animals cry with Polymorphism
The Cat speaks  : meow meow
The Dog speaks  : bow bow
The Cow speaks  : amba amba
From main   :  Cries are over
```

7.8.5 Abstract Classes

Abstract classes are classes that are partially implemented. Abstract method is a method that is declared, but not implemented. The general form is:

```
abstract type name(parameter-list);
```

For example:

```
abstract class AbstractClass
   {
   abstract void method1( );
   abstract int method2 (int x, int y);
   }
```

Here, `abstract` is a keyword. You cannot make constructor function `abstract`, `static` methods `abstract` and make `private` methods `abstract`.

`Abstract` classes are used as base classes for deriving subclasses. In the child class, all the `abstract` methods of the base class will be overridden.

7.9 FINAL VARIABLES

Java provides a keyword `final` that can be applied to variables, methods and classes. The meaning of the `final` keyword when applied to variables is different from its meaning when it is applied to methods and classes.

7.9.1 Final Variables

The value of a variable can be prevented from being modified by declaring a variable as final.

The syntax for declaring a final variable is:

```
final data_type variable_name=constant_value;
```

An attempt to alter the value of a final variable will generate a compile-time error. A final variable does not occupy memory for each instance of the class. Hence, final variable is a constant.

Example 11: A program to demonstrate the use of a final variable

```
class Person
{
```

```
    final int salary=30000;    // final variable
}
class FinalVariable
{
    public static void main(String args[])
    {
        Person obj=new Person();
        obj.salary=3000+2000;      // error
    }
}
```

NOTES

In this example, a compile-time error will be generated as the final variable `salary` cannot be modified.

7.9.2 Final Methods

As stated earlier, whenever a method is defined in the subclass with the same name and same signature as in the superclass, it is by default overridden in the subclass. However, a method can be prevented from being overridden by the subclass by declaring the method as final.

The syntax for declaring a final method is

```
final data_type method_name()
{
    :
}
```

Program 12: A program to demonstrate the use of final method

```
class Person
{
    String name="John";
    int age=25;
    final void result()     // final method
    {
        System.out.println("The name and the age of the
    person is: "+name+" and "+age);
    }
}
class Employee extends Person
{
    int salary=30000;
    void result()         //error
    {
        System.out.println("The salary of the employee is:
        "+ salary);
```

NOTES

```
    }
}
class FinalMethod
{
    public static void main(String args[])
    {
        Employee obj=new Employee();
        obj.result();
    }
}
```

This program will generate a compile-time error as the method `result()` is declared `final` and it cannot be overridden in the subclass.

7.9.3 Final Classes

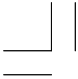
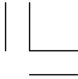
A class which cannot be further inherited or cannot have any subclass is known as final class. To make a class final, the class declaration is preceded by the `final` keyword. If a class is declared as `final`, all of its methods are automatically declared as `final`.

The syntax for declaring a final class is

```
final class class_name
{
    :
}
```

Program 13: A program to demonstrate the implementation of final class

```
final class Person  // final class
{
    String name="John";
    int age=25;
    void result()
    {
        System.out.println("The name and the age of the
    person are: "+name+" and "+age+"respectively");
    }
}
class Employee extends Person //error
{
    int salary=30000;
    void display()
```

NOTES

```
    {
       System.out.println("The salary of the employee is:
       "+salary);
    }
}
class FinalClass
{
    public static void main(String args[])
    {
       Employee obj=new Employee();
       obj.result();
       obj.display();
    }
}
```

This program will generate a compile-time error, since, the class `Person` is a final class and cannot be further inherited.

7.10 ABSTRACT METHODS AND CLASSES

A method can be prevented from being overridden in the subclass by using the `final` keyword. However, sometimes, there might be a situation, when the method in the superclass always needs to be redefined in the subclass and for this overriding becomes necessary. This kind of situation can arise when the superclass is not able to provide meaningful implementation for its methods. In such a case, the superclass provides a generalized structure of the method and leaves the implementation part to its subclass. To deal with such a situation, Java allows you to specify that a method must always be overridden in the subclass by using the keyword `abstract` in the method declaration.

The syntax for declaring an abstract method is:

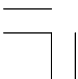
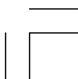
```
abstract data_type method_name();
```

If a class has one or more `abstract` methods, the class must also be declared as `abstract` by using the keyword `abstract`.

The syntax for declaring an `abstract` class is:

```
abstract class class_name
{
abstract data_type method_name();
}
```

The class which inherits the abstract class must provide implementation for all the methods or it should declare itself as `abstract`.

NOTES

These are certain points that should be kept in mind while using an `abstract` class, which are as follows:

- An `abstract` class cannot be instantiated.
- `Abstract` methods of an `abstract` class must always be implemented in the subclass.
- `Abstract` methods must end with semicolon `(;)` because they do not have any functionality.
- There can be both defined and undefined methods in an `abstract` class.
- Constructors or `static` methods cannot be declared as `abstract`.

Check Your Progress

7. Why an inner class cannot define any static member itself?
8. What is the use of super keyword?
9. Which keyword prevents a method in a superclass from being overridden by its subclass?
10. Does the abstract method contain only undefined methods?

7.11 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A class is a user defined data type that can be used to create instances of its type called objects.
2. The variables declared in a class are known as instance variables.
3. The two ways by which an argument can be passed to the method are call by value and call by reference.
4. A constructor is a special method which is used to initialize objects at the time of their creation.
5. A parameterized constructor is a constructor that accepts one or more parameters at the time of creation of objects and initializes the instance variables of the objects with these parameters.
6. The difference between the members of a class is that unlike non-static members, the static members are associated with the class as a whole, rather than with individual objects.
7. An inner class cannot define any static members itself because it is associated with an instance. Objects that are instances of an inner class exist within an instance of the outer class.

8. `Super` is a Java keyword used to reference non-static methods or variables of the parent class that may be hidden by the current class.
9. The keyword `final` prevents a superclass from being overridden by its subclass.
10. No, the abstract method cannot contain various methods other than the undefined methods.

NOTES

7.12 SUMMARY

- The key objective of object oriented programming is to represent the various real world objects as program elements.
- A class is a user defined data type that can be used to create instances of its type called objects.
- Methods are added to the class to access the instance variables of the class and manipulate them.
- Objects are variables of class type inside which they are created.
- The instance variables and methods added in the program cannot be accessed directly outside the class using their names. To access the variables and methods outside the class the dot (.) operator is used.
- The argument(s) that appear in the method call statement are known as actual arguments and those which appear in the method definition are known as formal arguments.
- A constructor is a special method which is used to initialize objects at the time of their creation.
- A parameterized constructor is a constructor that accepts one or more parameters at the time of creation of objects and initializes the instance variables of the objects with these parameters.
- Method overloading allows multiple methods to have the same name with different parameters and definitions.
- The `static` members of a class are defined without reference to a specific object. The `static` members are associated with the class as a whole, rather than with individual objects.
- Nested classes are divided into two categories: static and non-static. Nested classes that are declared `static` are simply called `static` nested classes. Non-static nested classes are called inner classes.
- If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such 'helper classes' makes the package more streamlined.

NOTES

- A `static` nested class cannot refer directly to instance variables or methods defined in its enclosing class like `static` class methods it can use them only through an object reference.
- An instance of `InnerClass` can exist only within an instance of `OuterClass` and has direct access to the methods and fields of its enclosing instance.
- Inheritance is an object oriented program (OOP) term in which the non-private features and attributes of a given superclass are made available to its subclass (es).
- When a superclass and subclass have the same method signature (including return type) for a method, then the method in the subclass is said to override the method of superclass. This involves changing the method body in the subclass.
- Overriding is useful to avoid new method names when the methods perform closely related functions under different conditions. With overriding you can provide different functionalities for the same methods. This leads to polymorphism and increases the program readability in big programs.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is determined at runtime instead at the time of compiling the program. This is how Java implements runtime polymorphism.
- Java provides a keyword `final` that can be applied to variables, methods and classes. Variables declared as `final` cannot be modified.
- A method can be prevented from being overridden by the subclass by declaring the method as `final`.
- A class which cannot be further inherited or cannot have any subclass is known as final class.
- The class which inherits the abstract class must provide implementation for all the methods or it should declare itself as `abstract`.

7.13 KEY WORDS

- **Class:** It is a user defined data type that can be used to create instances of its type called objects.
- **Instantiation:** The process of creating an object of a class.
- **Actual arguments:** The argument(s) that appear in the method call statement.
- **Constructor:** A special method which is used to initialize objects at the time of their creation.

7.14 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

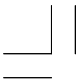
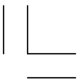
Short-Answer Questions

1. How is a class related to objects?
2. What is the difference between the two possible ways of passing arguments to a method?
3. State the difference between a method and a constructor.
4. Write a short note on the significance of method overloading.
5. What is the difference between method overloading and method overriding?
6. What is the use of nested classes?
7. How the method overriding is useful in OOPs?
8. How does inheritance facilitate code reusability?

Long-Answer Questions

1. Define a class `Student` having instance variables: `roll_number, marks1, marks2` and `marks3` and method: `calculate()` to calculate the overall percentage of marks and return the percentage of marks.
2. Write a program to display first ten terms of Fibonacci series.

   0 1 1 2 3 5 8……….
3. Can the objects of a class be initialized at runtime using constructors? Explain with an example.
4. Explain how you will identify the instance and class variables in a given class.
5. How is a `static` method invoked? Explain with an example.
6. Define a class `Teacher` having instance variables: `Teacher_id, Basic`, `DA` and `HRA` and method: `calculate()` to calculate the salary. (Hint: Salary is the sum of `Basic`, `DA` and `HRA`).
7. Explain multilevel hierarchy of inheritance with the help of example code.
8. Write a program to show how `super` keyword can be used to call superclass' constructor within a subclass' constructor.
9. Write a program to create a `Drug` class. A class `Tablet` is derived from it, which in turn is inherited by class `Pain Reliever`. Explain the type of inheritance implemented.

7.15 FURTHER READINGS

NOTES

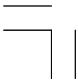
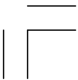
Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

UNIT 8 ARRAYS, STRINGS AND VECTORS

NOTES

Structure

8.0 INTRODUCTION

Handling real world data requires a mechanism that deals with a collection of data items. In Java, different data types like arrays, strings and vectors are offered to handle such collections. These data types are capable of grouping related data items together into a single unit. Arrays can hold a few or thousand data items of similar data types under a single name, strings are used to represent a collection of characters and vectors provide a way to keep items of any data type in any number, together as a single entity.

Fundamentally, an array is a block of container that can hold data of one single type. The array is a fundamental construct in Java that allows you to store and access a large number of values conveniently. Vector is like the dynamic array which can grow or shrink its size. Unlike array, we can store *n*-number of elements in it as there is no size limit. In Java, the strings are used for storing text.

In this unit, you will study about the basic concepts of arrays, one dimensional arrays, creating of array, two dimensional arrays, strings, vectors, wrapper classes, enumerated types and interfaces for multiple inheritance.

NOTES

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Declare, initialize and manipulate single- and multi-dimensional arrays
- Explain the use of `String` and `StringBuffer` classes for handling strings in Java
- Understand the significance of vectors
- Explain the concept of wrapper classes

8.2 ARRAYS

Java uses variables of different primitive data types to store data. However, these variables are incapable of holding more than one value at a time. For example, a single variable cannot be used for storing the marks of all the students in a class. For such purposes, Java provides a different kind of data type known as arrays.

An arrays is defined as a fixed-size sequence of the same type of data elements. These data elements can be of any primitive or non-primitive data type. The elements of an array are stored in contiguous memory locations and each individual element can be accessed using one or more indices or subscripts. A subscript or an index is a positive integer value, which indicates the position of an element in an array. Arrays are used when a programmer wants to store multiple data items of the same type into a single list and also wants to access and manipulate individual elements of the list. Arrays can be either single-dimensional or multi-dimensional depending upon the number of subscripts used.

8.2.1 Single-Dimensional Arrays

A single-dimensional array is the simplest form of an array that requires only one subscript to access an array element. Like an ordinary variable, an array must be declared before it is used in the program.

The syntax for declaring a single-dimensional array is

```
data_type array_name[];
```

or

```
data_type[]array_name;
```

where,

`data_type` is any data type

`array_name` is the name of the array

For example, an array `marks[]` of type `int` can be declared using either of the two statements.

```
int marks[];
```

or

```
int[] marks;
```

After an array is declared, we need to create it by allocating space to it in the memory. Arrays are created using the new operator.

The syntax for creating an array is

```
array_name=new data_type[size];
```

where,

`size` is the size of the array

For example, an array `marks[]` of type `int` and size five can be created using this statement.

```
marks=new int[5];
```

The given steps of declaration and creation of an array can be combined into a single statement as shown.

```
data_type array_name=new data_type[size];
```

Similarly, the statement to declare and create an array `marks[]` of type `int` and size five is

```
int marks[]=new int[5];
```

Note: All the elements created using the new operator in the array will be automatically initialized to zero.

Initialization of Single-Dimensional Array

Once an array is declared and memory is allocated to it, the next step is to initialize each array element with a valid and appropriate value. An array can be initialized at the time of its declaration.

The syntax for initializing an array at the time of its declaration is

```
data_type array_name[]={value_1,value_2,......,value_n};
```

The values are assigned to the array elements in the order in which they are listed. That is, `value_1`, `value_2` and `value_n` are assigned to the first, second and n^{th} element of the array, respectively.

If an array is declared and initialized simultaneously, then specifying its size is optional. For example, the statement `int marks[]={51,62,43,74,55}` is also valid. The size of an array `marks` can be obtained by using `marks.length()` method.

Note: If you try to store or access values outside the range of array (index with negative value or value greater than the length of the array), run-time error is generated**.**

NOTES

Accessing Single-Dimensional Array Elements

Once an array is declared and initialized, the values stored in the array can be accessed any time. Each individual array element can be accessed using the name of the array and the subscript value. Every element in an array is associated with a unique subscript value, starting from 0 to `size-1` (where, `size` refers to the maximum number of elements that can be stored in the array).

The syntax for accessing the values stored in a single-dimensional array is

```
array_name[subscript]
```

For example, the elements of the array `marks` can be referred to as `marks[0]`, `marks[1]`, `marks[2]`, `marks[3]` and `marks[4]`, respectively. Note that index of an array starts with `0`.

Note: The memory location, where the first element of an array is stored, is known as the base address, which is generally referred to by the name of the array.

Single-dimensional arrays are always allocated contiguous blocks of memory. This implies that every element in an array is always stored in a sequential manner next to each other. The memory representation of the array `marks` is shown in Figure 8.1. As each element is of type `int` (that is, 4 bytes long), the array `marks` occupies twenty contiguous bytes in the memory and these bytes are reserved in the memory at the compile-time.

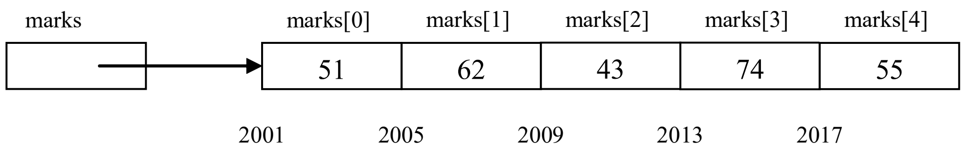


Fig. 8.1 *Memory Representation of an Array* `Marks`

Manipulation of Single-Dimensional Array Elements

An array can be manipulated with the help of various operations. These operations include finding the sum, average, maximum or minimum, sorting and searching of the array elements, etc.

Program 1: A program to sort the array elements

```
class SortingArray
{
       public static void main(String args[])
       {
          int a[]={67, 34, 12, 98, 26}; //array
       initialization at the time of declaration
          int n=a.length; //returns the length of the
                               array
          System.out.print("The list of numbers:");
```

NOTES

```
            for(int i=0;i<n;i++)
            {
               System.out.print(" "+a[i]);
            }
            //sorting elements of an array
            for(int i=0;i<n;i++)
            {
               for(int j=i+1;j<n;j++)
            {
               if(a[i]>a[j])
               {
                  int temp=a[i];
                  a[i]=a[j];
                  a[j]=temp;
               }
            }
         }
         System.out.print("\n");
         System.out.print("The sorted list of given
      numbers:");
         for(int i=0;i<n;i++) //displaying sorted array
         {
            System.out.print(" " +a[i]);
         }
      }
   }
```

Output of the program:

```
The list of numbers: 67 34 12 98 26
The sorted list of given numbers: 12 26 34 67 98
```

8.2.2 Two-Dimensional Array

A two-dimensional array is the simplest form of a multi-dimensional array that requires two subscript values to access an array element. Two-dimensional arrays are useful when the data being processed are to be arranged in the form of rows and columns (matrix form).

The syntax for declaring a two-dimensional array is

```
data_type array_name[][];
```

or

```
data_type[][] array_name;
```

NOTES

The syntax for creating a two dimensional array is

```
array_name[][]=new data_type[row_size][column_size];
```

The given steps of declaration and creation can be combined into one using a single statement as shown.

```
data_type  array_name[][]=newdata_type[row_size]
[column_size];
```

For example, an array `a[][]` of type `int` having three rows and two columns can be declared and created using this statement.

```
int a[][]=new int[3][2];
```

Here, 3 is the row size and 2 is the column size.

Initialization of Two-Dimensional Array

Like a single-dimensional array, a two-dimensional array can also be declared and initialized at the same time. To understand how to initialize a two-dimensional array, consider this statement.

```
int a[3][2]={  {101,51},
               {102,67},
               {103,76}   };
```

In this statement, an array `a[][]` of type `int` having three rows and two columns is declared and initialized. This type of initialization is generally used to increase the readability.

Now, consider another statement.

```
int b[][]={  { 2,3,4}, {1,1,1} };
```

In this statement, an array `b[][]` of type `int` having two rows and three columns is initialized.

Accessing Two-Dimensional Array Elements

Once a two-dimensional array is declared and initialized, the value stored in the array elements can be accessed using two subscripts.

The syntax for accessing the two-dimensional array elements is

```
array_name[row][column]
```

The first subscript value (`row`) specifies the row number and the second subscript value (`column`) specifies the column number. Both the subscript values specify the position of the array element within the array.

For example, the elements of array `a` (declared earlier) are referred to as `a[0][0]`, `a[0][1]`, `a[1][0]`, `a[1][1]`, `a[2][0]` and `a[2][1]` respectively.

NOTES

Generally, two-dimensional arrays are represented with the help of a matrix. However, in actual implementation, two-dimensional arrays are always allocated contiguous blocks of memory. Figure 8.2 shows matrix and memory representation of two-dimensional array a.

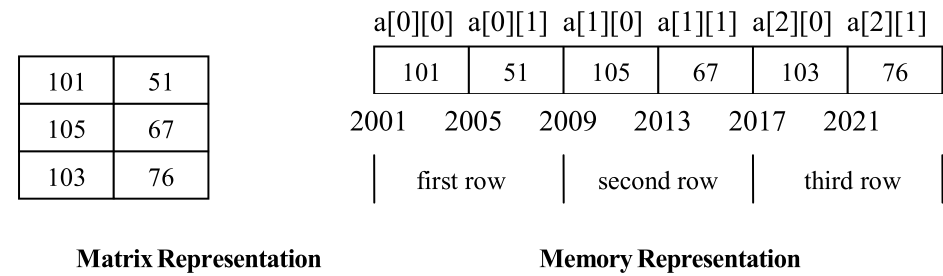


Fig. 8.2 *Matrix and Memory Representation of Array* a [ ] [ ]

Manipulation of Two-Dimensional Array Elements

A two-dimensional array can be manipulated in many ways. Some of the common operations that can be performed on a two-dimensional array include finding the sum of row elements, column elements and diagonal elements, finding the maximum and minimum values, etc.

Program 2: A program to calculate the sum of two matrices

```
class MatricesSummation
{
    public static void main(String args[])
    {
       int a[][]={{3,4,5},{3,2,7}};//initializing matrix a
       int b[][]={{2,4,7},{1,2,2}}; //initializing matrix b

       int l=a.length;
       System.out.println("First matrix is:" + " ");
       for(int i=0;i<l;i++) //displaying first matrix
       {
          for (int j=0;j<3; j++)
          {
             System.out.print(" " +a[i][j]);
          }
          System.out.println();
       }

       int m=b.length;
       System.out.println("Second matrix is:"+ " ");
       for (int i=0;i<m;i++) //displaying second matrix
```

NOTES

```
        {
           for(int j=0;j<3;j++)
           {
              System.out.print(" " + b[i][j]);
           }
           System.out.println();
        }

        System.out.println("Summation of the two matrices
        is: ");
        //displaying sum of two matrices
        for(int i=0;i<m;i++)
        {
           for(int j=0;j<=m;j++)
           {
              System.out.print(" "+(a[i][j]+b[i][j]));
           }
           System.out.println();
        }
     }
  }
```

Output of the program:

```
First matrix is:
 3 4 5
 3 2 7
Second matrix is:
 2 4 7
 1 2 2
Summation of the two matrices is:
 5 8 12
 4 4 9
```

Variable Size Arrays

In multi-dimensional arrays, the size of each array can be varied. For example, consider these statements.

```
data_type array_name[][]=new data_type[size][];
array_name[0]=new data_type[size_1];
.
.
.
```

```
array_name[n-1]=new data_type[size_n];
```

where,

size is the number of rows in two-dimensional array

size_1,..., size_n represents the number of columns in each row of two-dimensional array

NOTES

Check Your Progress

1. Define an array.
2. What is a subscript?

8.3 STRINGS

In Java, a string is an object that is created either using String or StringBuffer class. Each class has its own set of methods for creating and manipulating strings. The difference between these two classes is that a string created from the String class cannot be modified, that is, characters cannot be inserted to, replaced or removed from the string. However, the string created using StringBuffer class can not only be modified but can also be expanded or contracted dynamically, whenever required.

8.3.1 String Class

The String class is more commonly used to display messages and when strings need to be compared, searched or individual characters in a string have to be extracted as a substring.

The syntax to declare string is

```
String string_name;
```

The syntax for creating a string is

```
string_name=new String("Sequence_of_characters");
```

These two steps of declaration and creation can be combined into a single statement as shown.

```
String string_name=new String("Sequence_of_characters");
```

For example, the statement to declare and create a string str1 using String class is

```
String str1=new String("Java Programming Language");
```

The String class provides various methods for manipulating strings. Some of the most commonly used methods of String class along with their descriptions are listed in the Table 8.1.

NOTES

Table 8.1 String Class Methods and their Description

Methods	Description
`str1.length()`	Returns the length of the string `str1`
`str1.equals(str2)`	Returns '`true`' if string `str1` is equal to string `str2`
`str1.compareTo(str2)`	Returns negative if `str1<str2`, positive if `str1>str2`, otherwise `0`
`str1.concat(str2)`	Concatenates string `str1` and string `str2`
`str1=str2.trim()`	Removes all the white spaces at the beginning and end of the string `str2` and assigns it to `str1`
`str1=str2.replace('a','b')`	Replaces all `a` appearing in the string `str2` with `b` and assigns it to `str1`
`str1=str2.toLowerCase()`	Converts uppercase letters in a string `str2` to lowercase and assigns it to `str1`
`str1=str2.toUpperCase()`	Converts lowercase letters in a string `str2` to uppercase and assigns it to `str1`
`str1.indexOf('a')`	Gives the position of the first occurrence of character `'a'` in the string `str1`
`str1.indexOf('a',n)`	Gives the position of the first occurrence of character `'a'` that occurs after `n`th position in the string `str1`

Program 3: A program to demonstrate the use of some of the methods of a `String` class

```
class StringDemonstrate
{
    public static void main(String args[])
    {
        String str1=new String("New"); //creating string
                                str1
        String str2=new String("Delhi"); //creating string
                                str2
        String str3=str1.concat(str2); //concatenating
                    strings str1 and str2
        String str4=str3.toUpperCase();
        String str5=str3.toLowerCase();

        System.out.println("Combined String is: " +str3);
        System.out.println("Combined String in UPPER CASE
        is: " +str4);
        System.out.println("Combined String in LOWER Case
        is: " +str5);
    }
}
```

Output of the program:

```
Combined String is: NewDelhi
Combined String in UPPER CASE is: NEWDELHI
Combined String in LOWER Case is: newdelhi
```

8.3.2 StringBuffer Class

As stated earlier, strings created using `String` class are of fixed length, whereas, strings created using `StringBuffer` class are of varying length.

The syntax to declare string is

```
StringBuffer string_name;
```

The syntax for creating a string is

```
string_name=new StringBuffer ("Sequence_of_characters");
```

These two steps of declaration and creation can be combined into a single statement as shown below.

```
StringBuffer string_name=new StringBuffer
("Sequence_of_characters");
```

For example, the statement to declare and create a string `str1` using `StringBuffer` class is

```
StringBuffer str1=new StringBuffer("Java Programming
Language");
```

Some of the commonly used methods of `StringBuffer` class along with their descriptions are listed in Table 8.2.

Table 8.2 *`StringBuffer` Class Methods and their Description*

Methods	Description
`str1.append(str2)`	Appends the string `str2` at the end of the string `str1`
`str1.insert(n,str2)`	Inserts the string `str2` at the position n of the string `str1`
`str1.setCharAt(n,'x')`	Sets the n^{th} character of the string `str1` to `x`
`str1.setLength(n)`	Sets the length of the string `str1` to `n`. If `n<str1.length()`, `str1` is truncated. If `n>str1.length()`, null characters are added at the end of `str1`
`str1.reverse()`	Reverses the string `str1`
`str1.delete(m,n)`	Deletes characters of the string `str1` from m^{th} index to $(n-1)^{th}$ index
`str1.deleteCharAt(m)`	Removes character of the string `str1` at m^{th} index
`str1.replace(m,n,"str2")`	Replaces the portion of the string `str1` from m^{th} index to $(n-1)^{th}$ index with the string `str2`

NOTES

Check Your Progress

3. What is a string in Java?
4. State the difference in the strings created by `String` and `StringBuffer` class.

8.4 VECTORS

The `vector` class contained in the `java.util` package defines methods to store objects into a single unit. It can be used to implement a dynamic array called vectors which can accommodate number and any type of objects. Consider the following statements.

```
Vector v1=new Vector( ); //creating vector without
                specifying its initial capacity
Vector v2=new Vector(n); //creating vector having initial
                capacity 'n'
```

Here, the first statement creates a vector `v1` having initial capacity 10. That is, when you create a vector without specifying its initial capacity it is automatically set to 10. Similarly, the second statement creates a vector `v2` with initial capacity `n`. Although you have created the vector `v2` by specifying its initial capacity, this specification does not pose any limitation on the size of the vector since the size of the vector can be increased or decreased by adding or removing objects to it.

The `vector` class provides a variety of methods which can be used to perform different operations on vectors. Some of which are listed in Table 8.3.

Table 8.3 *Vector Methods and their Description*

Methods	Description
`vect1.addElement(object)`	Adds the specified `object` at the end of the vector list `vect1`
`vect1.size()`	Returns the number of objects currently present in the vector `vect1`
`vect1.capacity()`	Returns the maximum capacity of the vector `vect1`
`vect1.removeElement(object)`	Removes the specified object from the vector `vect1`
`vect1.elementAt(n)`	Returns the name of the n^{th} object of the vector `vect1`
`vect1.removeElementAt(n)`	Removes the item at the n^{th} position of the vector `vect1`
`vect1.removeAllElements()`	Removes all the elements in the vector `vect1`
`vect1.firstElement()`	Returns the first element of the vector `vect1`
`vect1.lastElement()`	Returns the last element of the vector `vect1`
`vect1.trimToSize()`	Sets the capacity of the vector `vect1` to the number of objects it is currently holding

8.5 WRAPPER CLASSES

NOTES

One limitation of using vectors is that we cannot store data items of primitive data types like `int`, `float`, `char`, `long`, `double` etc. since it can store only objects. Therefore, primitive data types need to be converted into objects before storing them in vectors. To accomplish this, Java provides wrapper classes contained in the `java.lang` package. The wrapper classes are used to convert the primitive data types into their respective class instances. For example, a `float` data type is represented as an instance of `Float` class.

It should be noted that Java's primitive wrapper classes are immutable, that is, once a value is assigned to an instance of a wrapper class, it cannot be changed later. The primitive data types and their corresponding wrapper classes are shown in Table 8.4.

Table 8.4 *Primitive Data Types and their Corresponding Wrapper Classes*

Primitive Data Type	Wrapper Class
`boolean`	java.lang.Boolean
`byte`	java.lang.Byte
`char`	java.lang.Character
`double`	java.lang.Double
`float`	java.lang.Float
`int`	java.lang.Integer
`long`	java.lang.Long
`short`	java.lang.Short
`void`	java.lang.Void

Some of the methods of wrapper classes are listed in Tables 8.5–8.9.

Table 8.5 *Conversion of Primitive Data Types to Object Types using Constructor Methods*

Constructor Calling	Action Performed
`Integer a=new Integer(i);`	Converts integer to Integer object
`Float b=new Float(f);`	Converts float to Float object
`Double c=new Double(d);`	Converts double to Double object
`Long d=new Long(l);`	Converts long to Long object

Table 8.6 *Conversion of Object Numbers to Primitive Numbers using* `typeValue()` *Method*

Method Calling	Action Performed
`int i=a.intValue();`	Converts object to Integer
`float f=b.floatValue();`	Converts object to Float
`double d=c.doubleValue();`	Converts object to Double
`long l=d.longValue();`	Converts object to Long

NOTES

Table 8.7 *Conversion of Primitive Numbers to Strings using* `toString()` *Method*

Method Calling	Action Performed
`str=Integer.toString(i);`	Converts Primitive integer to string
`str=Float.toString(f);`	Converts Primitive float to string
`str=Long.toString(l);`	Converts Primitive long to string
`str=Double.toString(d);`	Converts Primitive double to string

Table 8.8 *Conversion of String Objects to Numeric Objects using* `Valueof()` *Method*

Method Calling	Action Performed
`IntVal=Integer.valueOf(str);`	Converts string to Integer object
`FloatVal=Float.valueOf(str);`	Converts string to Float object
`LongVal=Long.valueOf(str);`	Converts string to Long object
`DoubleVal=Double.valueOf(str);`	Converts string to Double object

Table 8.9 *Conversion of Strings to Primitive Numbers using Parsing Methods*

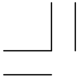
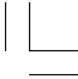
Method Calling	Action Performed
`int i=Integer.parseInt(str)`	Converts string to primitive integer
`long l=Long.parseLong(str)`	Converts string to primitive long

Note: The methods defined in the wrapper classes are static.

8.6 ENUMERATED TYPES

Enumerated means that a field has a set number of values. In Java, the enumerated or `enum` type is considered very significant method. Enumerations serve the purpose of representing a group of named constants in a programming language. Basically, the `enum` type is used when we know all the possible values at compile time, such as choices on a menu, rounding modes, command line flags, etc. The set of constants in an `enum` type may or may not stay fixed for all time. Consequently, enumerated means that something has a defined set of values it can hold. In Java, these values are stored in an enumerated variable, using the key word `enum`, which are unchanging and easily accessible.

In programming language, the values are static and final, which means that they cannot be changed. Other data types, such as `int` and `float`, can be changed in a program. Java key word `enum` are type safe, as no value can be assigned to key word `enum` which does not fit with the defined values. For example, the key word `enum` defined as 'DayOfWeek' will not accept any other data except the days of the week that were defined. In addition, an `enum` is a reference type, which means it acts more like a class or an interface. You can create methods and variables inside the `enum` declaration.

Syntax

The following syntax declares an enumerated (`enum`) variable for the days of the week.

```
public enum DayOfWeek {
SUNDAY,
MONDAY,
TUESDAY,
WEDNESDAY,
THURSDAY,
FRIDAY,
SATURDAY;
}
```

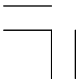
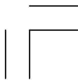
NOTES

Consider about the days of the week, it will always have Sunday through Saturday. We can assign a numeric value to the each day of the week, for example, Sunday will be Day (0), Monday will be Day (1), Tuesday will be Day (2), Wednesday will be Day (3), and so on. This can be declared in the Java program as follows.

```
public enum DayOfWeek {
SUNDAY (0),
MONDAY (1),
TUESDAY (2),
WEDNESDAY (3),
THURSDAY (4),
FRIDAY (5),
SATURDAY (6);
}
```

In the above code, we can create a constructor with a private variable so that it works appropriately. The constructor with a private variable can be declared within the `enum`, i.e., between the curly brackets that define the `enum`, as shown below:

```
public enum DayOfWeek {
SUNDAY (0),
MONDAY (1),
TUESDAY (2),
WEDNESDAY (3),
THURSDAY (4),
FRIDAY (5),
SATURDAY (6);
private int value;
private DayOfWeek(int value) {
```

```
this.value = value;
   }
}
```

NOTES

Thus by creating a variable we can access the values of the `enum`. To obtain the values of each part of the `enum`, we can declare the following public method:

```
public int getDayOfWeekValue() {
return this.value;
}
```

Thus the enumerated or `enum` data type in Java is used to describe a specific set of values for a variable. They are static or final and type safe which cannot be changed or modified. The `enum` types are reference type, meaning they act like a class and can have their own constructors and methods.

8.7 INTERFACES: MULTIPLE INHERITANCE

Object Oriented Programming (OOP) provides a user the feature of multiple inheritance, wherein a class can inherit the properties of more than a single parent class. Basically, multiple inheritance means that a class can be extended to more than one class. The programming language Java helps to use the multiple inheritance feature indirectly through the usage of interfaces.

Multiple inheritance is a feature of OOP concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority. An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class. Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.

In the following Java program example, we have two interfaces, the Motorbike and Cycle. Motorbike interface consists of the attribute speed. The method is `totalDistance()`. Cycle interface consists of the attribute `distance()` and the method `speed()`.

Both these interfaces are implemented by the class `TwoWheeler`.

Program 4: A Java program to demonstrate the use of two interfaces.

```
interface MotorBike
{
int speed=50;
public void totalDistance();
}
```

NOTES

```
interface Cycle
{
int distance=150;
public void speed();
}
public class TwoWheeler implements MotorBike,Cycle
{
int totalDistance;
int avgSpeed;
public void totalDistance()
{
totalDistance=speed*distance;
System.out.println("Total Distance Travelled :
"+totalDistance);
}
public void speed()
{
int avgSpeed=totalDistance/speed;
System.out.println("Average Speed maintained : "+avgSpeed);
}
public static void main(String args[])
{
TwoWheeler t1=new TwoWheeler();
t1.totalDistance();
t1.speed();
}
}
```

Output of the Program:

Total Distance Travelled : 7500

Average Speed Maintained : 150

The above Java Program 4 avoids ambiguity even when classes are used instead of interfaces. However, Java does not support it. When both the classes have the same method in it, the compiler is unable to decide upon the method to be called. Using interface avoids this ambiguity as the methods of interface are abstract by default.

The multiple inheritance in Java can be explained with the help of following Java program.

Program 5: A Java program to demonstrate the use of multiple inheritance without ambiguity.

NOTES

```
interface InterfaceOne
{
public void disp();
}
interface InterfaceTwo
{
public void disp();
}
public class Main implements InterfaceOne,InterfaceTwo
{
@Override
public void disp()
{
System.out.println("display() method implementation");
}
public static void main(String args[])
{
Main m = new Main();
m.disp();
}
}
```

The above Java Program 5 uses the `display()` method implementation. The `Main` method implements both the interfaces, i.e., `InterfaceOne` and `InterfaceTwo`. It executes without any ambiguity.

Check Your Progress

5. Why is the Vector class used?
6. What is the function of the wrapper classes?
7. In Java, why the enumerated or `enum` data type is used?
8. What is an interface? How an interface in Java supports multiple inheritance?

8.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Arrays are defined as a fixed size sequence of the same type of data elements. These data elements can be of any primitive or non-primitive data type.
2. A subscript is a positive integer value, which indicates the position of an element in an array.

3. In Java, a string is an object that is created either using `String` or `StringBuffer` class.
4. The strings created using String class are of fixed length whereas those created using `StringBuffer` class are of varying length.
5. The `Vector` class is used to implement a dynamic array called vector which can accommodate any number and any type of objects.
6. The wrapper classes are used to convert the primitive data types into their respective class instances.
7. The enumerated or `enum` data type in Java is used to describe a specific set of values for a variable. They are static or final and type safe which cannot be changed or modified. The `enum` types are reference type, meaning they act like a class and can have their own constructors and methods.
8. An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class. Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces. The programming language Java helps to use the multiple inheritance feature indirectly through the usage of interfaces.

8.9 SUMMARY

- An array is defined as a fixed-size sequence of the same type of data elements.
- The elements of an array are stored in contiguous memory locations and each individual element can be accessed using one or more indices or subscripts.
- Arrays can be either single-dimensional or multi-dimensional, depending upon the number of subscripts used.
- A single-dimensional array is the simplest form of an array that requires only one subscript to access an array element.
- Multi-dimensional arrays can be described as 'an array of arrays', that is, each element of the array is itself an array.
- A two-dimensional array is the simplest form of a multi-dimensional array that requires two subscript values to access an array element.
- A string is an object that is created either using `String` or `StringBuffer` class.
- The strings created using the `String` class are of fixed length and cannot be modified, that is, characters cannot be inserted to or removed from the string.

NOTES

- The strings created using the `StringBuffer` class are of varying length and can not only be modified but can also be expanded or contracted dynamically whenever required.
- The `vector` class is used to implement a dynamic array called vector which can accommodate any number and type of objects.
- The wrapper classes are used to convert the primitive data types into their respective class instances.
- Enumerated means that a field has a set number of values. In Java, the enumerated or `enum` type is considered very significant method.
- Enumerations serve the purpose of representing a group of named constants in a programming language.
- Basically, the `enum` type is used when we know all the possible values at compile time. In Java, these values are stored in an enumerated variable, using the key word `enum`, which are unchanging and easily accessible.
- Java key word `enum` are type safe, as no value can be assigned to key word `enum` which does not fit with the defined values.
- Object Oriented Programming (OOP) provides a user the feature of multiple inheritance, wherein a class can inherit the properties of more than a single parent class.
- Multiple inheritance means that a class can be extended to more than one class. The programming language Java helps to use the multiple inheritance feature indirectly through the usage of interfaces.
- An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class.
- Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.

8.10 KEY WORDS

- **Array:** It is a fixed size sequence of same type of data elements.
- **Subscript:** It is a positive integer value which indicates the position of an element in an array.
- **String:** It is an object that is created either using `String` or `StringBuffer` class.
- **Vector:** It is a dynamic array which can accommodate any number and type of objects.
- **`enum` data type:** The enumerated or `enum` data type in Java is used to describe a specific set of values for a variable.

- **Interface:** An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class.

8.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

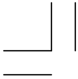
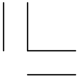
1. What is the need of arrays?
2. Differentiate between single-dimensional and multi-dimensional arrays.
3. What are vectors? How are they different from arrays?
4. What is the significance of enumerated types in Java?
5. Why multiple inheritance is used in Java?

Long-Answer Questions

1. Briefly discuss how the single-dimensional and multi-dimensional arrays can be declared, initialized and manipulated in Java. Give appropriate example Java programs.
2. Explain the use of `String` and `StringBuffer` classes for handling strings in Java with the help of appropriate Java programs.
3. Discuss the significance of vectors and wrapper classes in Java.
4. Briefly explain the significance of enumerated or `enum` data type in Java.
5. How the programming language Java helps to use the multiple inheritance feature through the usage of interfaces? Explain with the help of Java program.
6. Write a program to multiply two given matrices and print the resultant matrix.
7. Consider the given string:

```
String str="Java is an interesting programming
language.";
```

   (a) What will be the value of `str.length()`?
   (b) What will be the value of `str.charAt(12)`?
   (c) Write an expression to return letter e in the string `str`
8. Write a program to create a list of names of 5 students in a class and store them in a vector. In addition, perform the following operations in the list:
   (a) Add a name at the end of the list
   (b) Delete a name at the third position in the list
   (c) Display the final contents of the list

NOTES

8.12 FURTHER READINGS

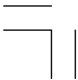
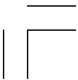
Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

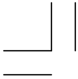
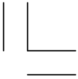
Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

UNIT 9 PACKAGES

NOTES

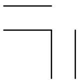
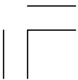
Structure

9.0 INTRODUCTION

Code reusability is one of the important features of object-oriented programming. So far we have seen that a code can be reused through inheritance. However, the reusability of a code through inheritance was restricted to only a single program. A situation may arise when the code of one program needs to be reused in some other program. This can be achieved in Java with the help of a concept known as **package**.

The concept of multiple inheritance is not directly supported by Java, i.e., a subclass in Java cannot have more than one superclass. However, there are certain applications where a class needs to inherit properties from multiple classes at the same time. Java provides an approach known as **interface** as a convenient alternative to implement multiple inheritance.

In this unit, you will study about Java Packages and Interface.

NOTES

9.1 OBJECTIVES

After going through this unit, you will be able to:

- Define package and its various advantages
- Explain various API packages and their application in a program
- Explain the method for creating and accessing a user-defined package
- Understand CLASSPATH
- Discuss the procedure for adding of class to a package and hiding a class in a package
- Differentiate between an interface and a class
- Describe the procedure for implementing an interface by a class
- Describe how an interface can be extended by another interface
- Explain how interfaces can be used to declare a set of constants
- State the differences between interface and abstract class

9.2 PACKAGES AND JAVA API PACKAGES

A package is a named collection of classes. Any number of related classes can be grouped into a single package. By grouping the classes into a package, we can achieve the following:

- The classes which belong to a package of another program can be easily reused.
- Two classes in two different packages can have the same name. For instance, we can create a class called Student without being concerned about the collision of this name with some other class called Student which belongs to some other package.

Every class belongs to a package. The classes that we have created and used so far belong to the **default** package. Java packages are categorized into two types, namely, *API packages* and *user-defined packages*. API (Application Programming Interface) packages are the standard packages available in Java which contain all of the standard classes. Java also allows us to create our own package known as *user-defined packages*.

Java API Packages

There is a huge list of API packages provided by Java. Some of the quite frequently used packages are displayed in Table 9.1.

Table 9.1 Commonly Used API Packages

Package	Description
java.lang	It includes classes that are fundamental to Java such as String, Math, Exception, etc.
java.io	This package provides classes to support input and output operations.
java.awt	This provides a set of classes to implement graphical user interface components such as windows, dialog boxes, menus, list, buttons, checkboxes, textfields, scrollbars, etc.
java.applet	This package includes classes to create applets that can be embedded in a webpage.
java.util	This package includes language utility classes such as time, date, random number, hash tables, vectors, enumeration, etc.
java.sql	This package contains classes for accessing database using standard SQL.

NOTES

Each Java API package is organized in a hierarchical form. Consider a package, for instance, java.lang. The java.lang package containing various classes is organized in a hierarchical form as shown in Figure 9.1.

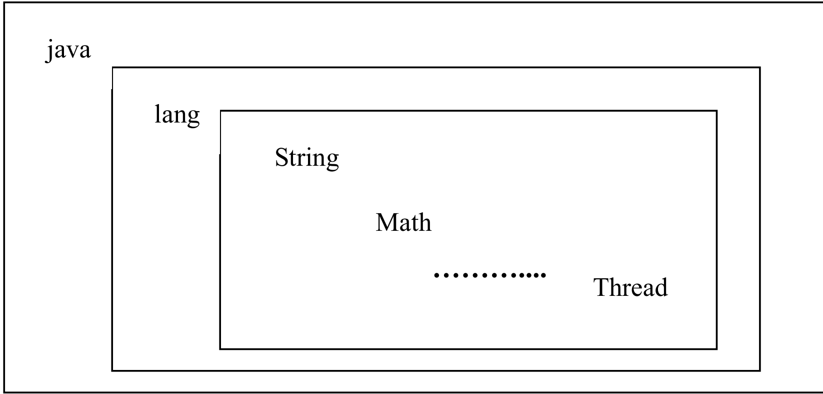


Fig. 9.1 Hierarchical Representation of java.lang Package

Figure 9.1 shows that the package java contains the package lang, which in turn contains various classes, such as Math, String, etc.

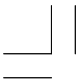
The classes stored in a package can be accessed in two ways depending on the following two situations:

- When we need to access the class only once or when we do not need to access any other class of the same package.

  In this case, to access the class, the name of the package is followed by dot (.) operator, which is followed by the name of the class. For instance, the statement to access the Math class contained in lang package is as follows:

  ```
  java.lang.Math
  ```

- If we want to use multiple classes contained in a package or use the same class in various places of the program.




NOTES

This can be achieved with the help of the import statement. The syntax to access the same class in various places is as follows:

```
import package_name.class_name;
```

where,

`import` is a Java keyword
`package_name` is the name of the package
`class_name` is the name of the class belonging to that package.

For instance, consider the following statement:

```
import java.lang.Math;
```

In this statement, the class Math contained in the package java.lang is imported in the program and can now be used at various places in the program without using the package name.

We can access multiple classes contained in a package using the following statement:

```
import package_name.*;
```

For instance, the statement import `java.lang.*` imports all the classes contained in the `java.lang` package.

Package Naming

There are certain Java naming conventions using which packages can be named. The names of the packages and classes or interfaces should be such that it is easier to distinguish between the two. According to the conventions, the name of the packages starts with a lowercase letter whereas the name of the classes begins with an uppercase letter. For instance, consider the following statement:

```
java.lang.Math
```

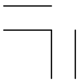
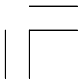
In this statement, the name of the package lang begins with a lowercase letter and the name of the class Math begins with an uppercase letter. However, as there are number of users working simultaneously on the Internet, there may be a chance of duplicate packages being created which may lead to run-time error. Hence, package names must be unique. To ensure uniqueness of the package names, the name of the package can be preceded with the Internet domain name. For instance, consider the following statement:

```
dpe.aet.package_name
```

In this statement, the package `package_name` is preceded by the domain name `dpe.aet`.

9.3 DEFINING PACKAGES

To create a user-defined package, first a package must be declared using the keyword package followed by the name of the package.

NOTES

The syntax to declare a package is as follows:

```
package package_name;
```

where ,

`package` is the Java keyword

`package_name` is the name of the package.

Note: *This statement must be the first statement in the Java source file.*

Once the package is declared, we can define any number of classes that will be a part of this package. For instance, consider the following code segment:

```
package packagename;
public class ClassName
{
//body of the class
}
```

The `ClassName` is the name of the class that will belong to the package `packagename`. In Java, packages are stored in the file system directories. The above code must be saved with the filename `ClassName.java` in a directory named `packagename`. When the Java compiler compiles the source file, the `ClassName.class` file is created and automatically stored in the same directory `packagename`.

Note: *Since Java is case sensitive, the name of the directory must exactly match with the package name.*

Like API packages, user-defined packages can be organized in a hierarchical structure. The package names forming part of this structure are separated by using dot (.) operator. For instance, consider this statement.

```
package mypackage1.mypackage2;
```

This package must be stored in `mypackage1/mypackage2` directory. By using multileveled package statement, the packages which are related to each other can be grouped into a single package.

Note: *If a Java package contains multiple classes, only one of them can be declared as `public` and the source file is saved with the name of the public class having `.java` extension.*

When a source file having multiple class definitions is compiled, the compiler creates separate `.class` file for each class.

Program 1: A program to demonstrate the creation of a package.

```
package mypackage;
class AreaRectangle
{
int  length , breadth;
    AreaRectangle(int l, int b)
    {
```

NOTES

```
length=l;
       breadth=b;
       int area=l*b;
System.out.println("The area of rectangle is: " +area);
    }
}
class SimplePackage
{
public static void main(String[] args)
    {
       AreaRectangle obj=new AreaRectangle(20,30);
    }
}
```

Save this file as `SimplePackage.java` under the directory called `mypackage` and compile this file. Now the file can be executed using the command line as follows:

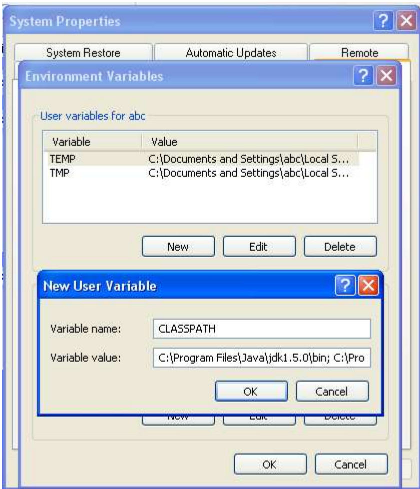
```
java mypackage.SimplePackage
```

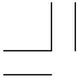
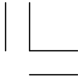
We cannot execute it using the command line `java SimplePackage` because `SimplePackage` is now a part of the `mypackage` package.

9.4 CLASSPATH

As packages in Java are stored in the file system directories, the Java run-time system needs to know where to look for the package that we create. There are two possible ways to locate a package. First, the Java run-time system by default uses the current working directory. Second, a directory path or paths can be specified by setting the **CLASSPATH** environmental variable.

The procedure required in order to set a CLASSPATH environmental variable is as follows:



NOTES

1. Right click on **My Computer** icon and then select **Properties** from the shortcut menu that appears. This displays the **System Properties** dialog box with **General** as the active tab.
2. Click **Advanced** tab.
3. Click the **Environment Variables** button. This displays the **Environmental Variables** dialog box.
4. Click the **New** button in the **User Variables** section. This displays the **New User Variable** dialog box as displayed in the screenshot.
5. Enter CLASSPATH as the variable name.
6. Enter the list of directories in your CLASSPATH as the value of the variable with a semicolon to separate entries it the list.
7. Click OK to close the **New User Variable** dialog box.
8. Click OK to close the **Environment Variables** dialog box.
9. Click OK to close the **System Properties** dialog box.

9.5 ACCESSIBILITY OF PACKAGES USING PACKAGE MEMBERS

The user-defined packages are accessed in the same way as Java standard packages. For instance, to access a class `MyClassName` of a package named `mypackage1, the` following statements can be used:

```
import mypackage1.MyClassName;  //statement1
```

or

```
import mypackage1.*;          //statement2
```

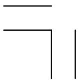
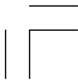
Thus, if statement 1 is used, all the members of `MyClassName` can be directly accessed without having to use the package name anywhere in the program. The asterisk (`*`) in statement 2 specifies that whenever the compiler has to find any class, it should search it in the package `mypackage1`. This allows all the classes of this package to be accessed directly. The main advantage of this approach is that the long package names need not be used repeatedly in the program. However, the disadvantage is that it becomes difficult to identify the package to which a particular member belongs.

Using Package Members

Once the required package is imported in the program, the classes belonging to that package can be used. To understand the concept of using the package, let us first create a package (`mypackagename1)`.

Program 2: A program to create mypackagename1.

```
package mypackagename1;
public class MyClassName1
{
```

```
public void firstresult()
{
System.out.println("This is the first class's result from
the first  package");
}
}
```

NOTES

Save the source file as `MyClassName1.java` in the subdirectory `mypackagename1`. After compiling this file, the corresponding compiled file `MyClassName1.class` will be stored in the same directory.

Program 3: A program to demonstrate the use of package `mypackagename1`.

```
import mypackagename1.MyClassName1;
class UsePackage
{
public static void main(String args[])
{
MyClassName1 obj=new MyClassName1();
obj.firstresult();
}
}
```

Output of the program:

```
This is the first class's result from the first package
```

Now, consider another package; `mypackagename2`.

Program 4: A program to create `mypackagename2`.

```
package mypackagename2;
public class MyClassName2
{
    public void secondresult()
    {
System.out.println("This is the second class's result
from the second package");
    }
 }
```

Program 5: A program to demonstrate importing multiple packages.

```
import  mypackagename1.MyClassName1;  //importing
MyClassName1
import  mypackagename2.MyClassName2;  //importing
MyClassName2
class UseMultiplePackage
{
   public static void main(String[] args)
    {
```

```
        MyClassName1 obj1=new MyClassName1();
        MyClassName2 obj2=new MyClassName2();
        obj1.firstresult();
    obj2.secondresult();
    }
}
```

NOTES

Output of the program:

```
This is the first class's result from the first package
This is the second class's result from the second package
```

When multiple packages are imported, there is a chance of having more than one package containing classes with the same name. For instance, consider the definitions of two packages `packagename1` and `packagename2` having the same class name `FirstClass.`

Program 6: A program to create `packagename1`.

```
package packagename1;
public class FirstClass
{
    int i=10;
    int j=20;
    public void display1()
     {
System.out.println("The value of i and j in the first
package is: "+i+" and "+j);
     }
}
```

Program 7: A program to create a `packagename2`.

```
package packagename2;
public class FirstClass
{
    float p=3.4F;
    float q=2.55F;
    public void display2()
     {
System.out.println("The value of p and q in the second
package is: "+p+" and "+q);
     }
}
```

Since both the packages contain the class `FirstClass`, the compiler will not be able to decide which package to use, thereby causing an ambiguity. In such a case, the class name must be preceded with the name of the package while creating the objects of the concerned class.

NOTES

Program 8: A program to demonstrate importing multiple packages having classes with same name.

```
import packagename1.*;
import packagename2.*;
class SameClassName
{
   public static void main(String[] args)
    {
packagename1.FirstClass              obj1=new
packagename1.FirstClass();
    obj1.display1();
packagename2.FirstClass              obj2=new
packagename2.FirstClass();
    obj2.display2();
    }
}
```

Output of the program:

```
The value of i and j in the first package is: 10 and 20
The value of p and q in the second package is: 3.4 and 2.55
```

In this illustration, the class `FirstClass` in the package `packagename1` has the same name as the class in the package `packagename2`. While creating the objects of the classes of each package, the class name is preceded by their respective package names, thereby causing no ambiguity.

Check Your Progress

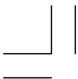
1. What is a package?
2. Write the statement for declaring a package named mypackage.
3. Which Java API package is automatically imported?
4. What is the use of import statement in Java?

9.6 ADDING A CLASS

A new class can be added to an already existing package. For instance, consider a package called `mypackage` whose definition is as follows:

```
package mypackage;
public class MyClass1
{
// body of MyClass1
}
```

The package `mypackage` contains a public class `MyClass1`. Now, suppose we want to add another class `MyClass2` to this package. A package cannot have




NOTES

more than one public class. If MyClass2 is non-public, then we can simply add its definition to the same source file as follows:

```
package mypackage;
public class MyClass1  //existing class
{
// body of MyClass1
}
class MyClass2        //new class
{
// body of MyClass2
}
```

Now recompile this source file. The MyClass2.class file is created and stored automatically in the package mypackage. Thus, mypackage contains two classes MyClass1 and MyClass2.

However, if the class MyClass2 is a public class, we need to create this class in a separate source file and declare the package statement at the top of the source file as follows:

```
package mypackage;
public class MyClass2
{
// body of MyClass2
}
```

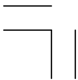
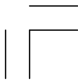
Compile this source file. The package mypackage will now contain MyClass2.class file as well. Thus, if we want to create multiple public classes in a package, we need to create a separate source file for each class and compile them. After compilation, the package will contain the .class files of all the source files.

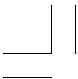
9.7 HIDING A CLASS

All the classes in a package which are declared public are automatically imported, when a package is imported using asterisk (*). However, most of the time, we like to prevent some classes from being accessed outside the package. To hide such classes, they should not be declared as public. For instance, consider a package packagename.

Program 9: A program to create packagename.

```
package packagename;
public class FirstClass //accessible to all
{
    int i;
    int j;
public FirstClass(int a ,int b)
```




NOTES

```
{
       i=a;
       j=b;
System.out.println("The value of i and j is:"+i+" and
"+j);
}
}
class SecondClass  //not public, hidden
{
    int p;
    int q;
    SecondClass(int x,int y)
    {
       p=x;
       q=y;
System.out.println("The value of p and q is:"+p+" and
"+q);
    }
}
```

The package `packagename` contains two classes, `FirstClass`, which is `public` and a `SecondClass`, which is non-public. Since, the `SecondClass` is not declared as `public`, it is hidden from the members outside the package `packagename`. It can be accessed only by the classes which are inside the same package `packagename`.

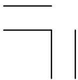
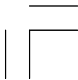
Program 10: A program to demonstrate the hiding of a class in a package.

```
import packagename.*;
class CallingClass
{
public static void main(String[] args)
    {
       FirstClass obj1=new FirstClass(20,30);
       SecondClass obj2=new SecondClass(30,40);     //
error
    }
}
```

In this illustration, a compile-time error will be generated as the class `SecondClass` is not public and cannot be imported.

9.8 PACKAGES AND VISIBILITY CONTROLS

There are three main access specifiers, namely `private`, `public` and `protected` which provide various access levels. It is simplified as follows for easy reference.

NOTES

Anything which is declared as `private` can be accessed only within the class. When a member is declared as `public`, it can be accessed everywhere. A member declared as `protected` is accessible not only to all the classes and subclasses in the same package but also to subclasses in other packages. However, a protected member cannot be accessed by the non-subclasses of other packages. If no access specifier is specified, by default, the data member of a class is visible only in the classes of the same package.

To understand the concept of visibility controls with respect to package, let us consider two packages `pkg1` and `pkg2`.

Program 11: A program to create a package `pkg1`.

```
package pkg1;
public class CommonClass  //superclass
{
    int i=1;
    private int j=2;
    protected int k=3;
    public int l=4;
    int sum=0;
    public CommonClass()
    {
       System.out.println("Common Class Constructor");
       System.out.println(" i = " +i);
       System.out.println(" j = " +j);
       System.out.println(" k = " +k);
       System.out.println(" l = " +l);
       sum = (i+j+k+l);
System.out.println("The summation of all members in
CommonClass is :" +sum);
    }
}
class NextClass extends CommonClass  //subclass
{
    NextClass()
    {
       System.out.println("Next Class Constructor");
       System.out.println(" i = " +i);
//System.out.println(" j = " +j);
System.out.println(" k = " +k);
       System.out.println(" l = " +l);
       sum = (i+k+l);
System.out.println("The summation of all members in
NextClass is :" +sum);
```

```
    }
}
```

Program 12: A program to demonstrate the accessibility of members in the package `pkg1`.

NOTES

```
package pkg1;
class FinalClasses
{
    public static void main(String args[])
    {
       NextClass obj1=new NextClass();
    }
}
```

Output of the program:

```
Common Class Constructor
 i = 1
 j = 2
 k = 3
 l = 4
The summation of all members in CommonClass is :10
Next Class Constructor
 i = 1
 k = 3
 l = 4
The summation of all members in NextClass is :8
```

All the variables except the `private` variable `j` can be accessed in the subclass `NextClass`.

Program 13: A program to create a package `pkg2`.

```
package pkg2;
class SecondCommonClass extends pkg1.CommonClass
{
    SecondCommonClass()
    {
       System.out.println("Second Class Constructor");
/*i declared as default in pkg1, so cannot be accessed in
pkg2*/
//System.out.println(" i = " +i);

/*j declared as private in pkg1, so cannot be accessed in
pkg2*/
//System.out.println(" j = " +j);
```

```
        System.out.println(" k = " +k);
        System.out.println(" l = " +l);
    }
}
```

NOTES

Program 14: A program to demonstrate the accessibility of members of `pkg1` in `pkg2`.

```
package pkg2;
class PClasses
{
    public static void main(String args[])
    {
        SecondCommonClass obj2 = new SecondCommonClass();
}
}
```

Output of the program:

```
Common Class Constructor
 i = 1
 j = 2
 k = 3
 l = 4
The summation of all members in CommonClass is: 10
Second Class Constructor
k = 3
l = 4
```

The class `SecondCommonClass` of package `pkg2` is the subclass of the class `CommonClass` of the first package `pkg1`. The private variable `j` and default variable `i` cannot be accessed in `pkg2` as they belong to package `pkg1`. However, the protected variable `k` can be accessed in `pkg2` since `SecondCommonClass` is the subclass of the class in package `pkg1`. The public variable `l` in any case can be accessed everywhere.

Check Your Progress

5. How a new class is added to an existing package?
6. How will you add a public class to a package that already contains another public class?

9.9 INTERFACE

An interface is quite similar to a class. The only difference is that it contains only final variables and method declarations. Hence, an interface can be considered as a

'fully abstract class'. There is no limitation to the number of interfaces that a class can implement. An interface is defined like a class but instead of the keyword `class`, the keyword `interface` is used.

NOTES

The syntax to define an interface is as follows:

```
interface interface_name
{
//variables and methods declaration
}
```

where,

`interface` is the Java keyword

`interface_name` is the name of the interface

If there is no access specifier included in the interface definition, the default access is used and the interface is visible only to the members of the same package. However, to make the interface accessible in any other code, it can be declared as `public`. The variables in an interface are by default `static` and `final`. Hence, they cannot be altered by the implementing class. The methods are by default `abstract`. All the methods must be implemented by the class which implements the interface. For instance, consider the following code segment:

```
interface Area
{
double pi=3.142;
void compute();
}
```

The term `Area` is the name of the interface. The variable `pi` is initialized with a constant value. It should be noted that the method `compute()` does not have the body part and its declaration ends with a semicolon.

Note: *If the interface is declared as public then, all the variables and methods are implicitly public.*

9.9.1 Implementing Interface

Once an interface is defined, it can be used as a superclass whose members and properties can be inherited by other classes. One or more classes can implement the interface by using the keyword `implements` in the class definition.

The syntax for implementing an interface is as follows:

```
class class_name implements interface_name
{
//variables and methods declarartion
}
```

For instance, consider the following code segment which implements the interface `Area`.

```
class Circle implements Area
{
float r=4.3F;
    public void compute()
    {
       double carea=pi*r*r;
       System.out.println("The area of circle is: "
+carea);
    }
}
```

NOTES

When the methods in an interface are defined in the implementing class, the `public` keyword must be used. Also, the signature of the method implementing the interface must exactly match the signature of the method declaration in the interface.

A class can implement more than one interface as displayed in the following example:

```
class class_name implements interface1, interface2
{
:
}
```

A class can extend another class while implementing interfaces as displayed in the following example:

```
class class_name extends superclass implements
interface_name
{
:
}
```

Program 15: A program to demonstrate implementation of interface.

```
interface Area  //interface
{
    double pi=3.142;
    void compute();
}
class Circle implements Area //implementing interface
{
    float r=4.3F;
    public void compute()  //defining interface's method
{
       double carea=pi*r*r;
       System.out.println("The area of circle is: "
+carea);
    }
```

NOTES

```
}
class Sphere implements Area //implementing interface
{
    float r=3.5F;
    public void compute() //defining interface's method
    {
       double sarea =4*pi*r*r;
       System.out.println("The  area  of  sphere  is:  "
+sarea);
    }
}
class ImplementInterface
{
    public static void main(String args[])
    {
       Circle cobj=new Circle();
       Sphere sobj=new Sphere();
       Area intobj;

       intobj=cobj;  //intobj refers to cobj
       intobj.compute();

       intobj=sobj;  //intobj refers to sobj
       intobj.compute();
    }
}
```

Output of the program:

```
The area of circle is: 58.095585153885004
The area of sphere is: 153.958
```

In this illustration, the interface `Area` is implemented by two classes `Circle` and `Sphere`. The method `compute()` of the interface is defined in both the classes but in different ways. A reference variable `intobj` is declared which is of type `Area` and is assigned the instances of `Circle` and `Sphere`. The interface reference variable `intobj` can be used to access the method `compute()` but it cannot be used to access any other members of the classes `Circle` and `Sphere` as it has knowledge of only those methods which are declared by its interface.

It should be noted that the method `compute()` can also be called through the reference variables of class type, i.e., through `cobj` and `sobj`. In this case, both the classes need to be present at compile-time. On the other hand, when we call a method through interface reference, the method to be executed is resolved

dynamically at run-time. This dynamic method resolution at run-time is one of the key features of interface.

Partial Implementations

NOTES

If a class that implements the interface does not provide complete implementation of the methods declared in the interface, then it is necessary for the class to be declared as an `abstract`.

Program 16: Consider a class Square which implements the interface `Area`.

```
abstract class Square implements Area
{
float side=2.4;
double sqarea=side*side;
void display()
{
   System.out.println("The area of square is: "+sqarea);
}
}
```

The class `Square` is declared as `abstract` as it does not implement the method `compute()` declared in `Area`. Any class that inherits `Square` must implement `compute()` method or the class itself must be declared as `abstract`.

9.9.2 Extending Interfaces

An interface can inherit another interface by using the `extends` keyword in the same way as a class inherits from another class. Like a class, a subinterface will inherit all the properties of the superinterface and also add its own data members. For instance, consider the following code segment.

```
interface  Interface1
{
    :
}
interface  Interface2 extends  Interface1
{
:
}
```

An interface can also inherit from more than one interface. To define an interface that extends several interfaces, the names of the superinterfaces are separated by a comma (,) as displayed here.

```
interface  Interface3  extends  Interface1, Interface2
{
:
}
```

NOTES

It should be noted that the methods declared in the superinterfaces cannot be implemented by the subinterfaces. They must be implemented only by the class which implements the interface. When a class implements an interface which is inherited from another interface then the class must provide implementation for all the methods declared in both the interfaces.

Program 17: A program to demonstrate extending of interface by another interface.

```
interface FirstInterface

{
    int i=30;
    void display1();
}
interface SecondInterface extends FirstInterface
{
    int j=i+40;
    void display2();
}
class ClassName implements SecondInterface
{
    public void display1()
    {
System.out.println("The value of i is: "+i);
    }
    public void display2()
    {
System.out.println("The value of j is: "+j);
    }
}
public class ExtendingInterface
{
    public static void main(String[] args)
    {
ClassName obj=new ClassName();
       obj.display1();
       obj.display2();
}
}
```

Output of the program:

```
The value of i is: 30
The value of j is: 70
```

In this illustration, the class `ClassName` implements the interface `SecondInterface`, which is inherited from the interface `FirstInterface`. Hence, `ClassName` provides implementation of both the methods `display1()` and `display2()` of the `FirstInterface` and the `SecondInterface` respectively.

Note: *An interface cannot extend classes. It can only extend another interface. Also, an interface cannot implement another interface.*

9.9.3 Variables in Interfaces

An interface can be used to declare a set of constants that can be used by the classes implementing the interface. It is same as creating header files in C/C++ which contains a large number of constants.

Program 18: A program to demonstrate implementation of interface having only constant values.

```
interface SetOfConstants
{
    int red=0;
    int green=1;
    int blue=2;
}

class UsingConstants implements SetOfConstants
{
    void display()
    {
       int color=2;
       switch(color)
       {
          case red:
             System.out.println("The color is red");
                break;
          case green:
             System.out.println("The color is green");
                break;
          case blue:
             System.out.println("The color is blue");
                break;
       }
    }
}
```

```
public class InterfaceConstants
 {
   public static void main(String[] args)
    {
       UsingConstants obj=new UsingConstants();
       obj.display();

    }
}
```

NOTES

Output of the program:

```
The color is blue
```

9.9.4 Interface and Abstract Classes

Interface and abstract class are similar in the way that neither interface nor abstract class can be instantiated. However, there are certain differences between them, which are as follows:

- An interface can contain only final members and abstract methods, whereas an abstract class can contain non-final and final members and both abstract and concrete methods. Thus, it can be concluded that an interface is a fully abstract class.
- While an interface is defined by using the keyword `interface`, an abstract class is defined using the `abstract` keyword.
- All the methods in an interface must be implemented by the class which implements the interface, whereas the methods in the abstract class need not be implemented by the class that extends the abstract class. If the class which extends the abstract class does not implement the methods, then the class itself must be declared as `abstract`.
- A class can inherit only one abstract class whereas a class can implement more than one interface.
- Members in an interface are by default public, whereas members in an abstract class can be private as well as protected.

9.9.5 Extends and Implements Together

By now you are familiar with the concept of interface. Now let us have a look at how interface can be used to implement multiple inheritance by taking a simple example. In this illustration, the class `Faculty` extends a class `Employee` and implements an interface `Bonus`.

Program 19: A program to demonstrate implementation of multiple inheritance through interface.

```
class Person
{
```

```
    String name;
    int age;
    String address;
    void persondetails(String nm, int ag, String add)
    {
       name=nm;
       age=ag;
       address=add;
    }
    void displayperson()
    {
       System.out.println("Name: "+name);
       System.out.println("Age: "+age);
       System.out.println("Address: "+address);
}
}
class Employee extends Person
{
    int empid;
    int salary;
    void empdetails(int id,int sal)
    {
empid=id;
       salary=sal;
    }
    void displayemployee()
    {
       System.out.println("Empid: "+empid);
       System.out.println("Salary: "+salary);
    }
}
interface Bonus
{
    int bonus=1000;
    void compute();
}
class Faculty extends Employee implements Bonus
{
    int amount;
    public void compute()
```

NOTES

NOTES

```
    {
       System.out.println("The bonus is: "+bonus);
       amount=salary+bonus;
    }
    void  facultydetails()
    {
       displayperson();
       displayemployee();
       compute();
       System.out.println("The total amount is: "+amount);
}
}
public class MultipleInheritance
{
public static void main(String[] args)
{
       Faculty obj=new Faculty();
obj.persondetails("Surabhi",23,"115,Greenfield Apartment.
Patparganj,New Delhi-110092");
       obj.empdetails(001,20000);
       obj.facultydetails();

       System.out.println("");

obj.persondetails("Mili",27,"D-50,Old Gupta Colony,Delhi-
110009");

obj.empdetails(002,30000);
       obj.facultydetails();
}
}
```

Output of the program:

```
Name: Surabhi
Age: 23
Address: 115, Greenfield Apartment, Patparganj,New Delhi-
110092
Empid: 1
Salary: 20000
The bonus is: 1000
The total amount is: 21000
```

```
Name: Mili
Age: 27
Address: D-50,Old Gupta Colony,Delhi-110009
Empid: 2
Salary: 30000
The bonus is: 1000
The total amount is: 31000
```

NOTES

Check Your Progress

7. How does an interface differ from a class?
8. What is the difference between `implements` and `extends` keyword?

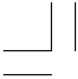
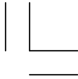
9.10 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A package is a named collection of classes.
2. The statement for declaring a package named `mypackage` is package mypackage;
3. `java.lang` package is automatically imported.
4. The `import` statement can be used to access a particular package in a program.
5. A new class can be added to an already existing package. For instance, consider a package called mypackage whose definition is as follows:
   ```
   package mypackage;
   public class MyClass1
   {
   // body of MyClass1
   }
   ```
6. We can add a public class to a package that already contains another public class by creating a separate source file for the class to be added and declaring the package statement at the top of the file.
7. Unlike a class, an interface contains only final variables and method declarations.
8. The `implements` keyword is used by the class to implement an interface. The `extends` keyword is used by the class to inherit another class and it is used by the interface to inherit another interface.

9.11 SUMMARY

NOTES

- A package is a named collection of classes. Any number of related classes can be grouped into a single package. The classes which belong to a package of another program can be easily reused. Two classes in two different packages can have the same name.
- Java packages are categorized into two types, namely, API packages and user-defined packages. API (Application Programming Interface) packages are the standard packages available in Java which contain all the standard classes. Java also allows the users to create their own packages known as user-defined packages.
- Popular packages include `java.lang, java.io, java.awt, java.applet, java.util` and `java.sql`.
- There are certain Java naming conventions that can be used for naming packages. The names of the packages and classes or interfaces should be such that it is easier to distinguish between the two. According to the conventions, names of packages start with a lowercase letter, whereas names of classes begin with an uppercase letter.
- To create a user-defined package, a package must be first declared using the keyword package followed by the name of the `package`. The package statement must be the first statement in the Java source file. Once the package is declared, we can define any number of classes which will be a part of the package.
- As packages in Java are stored in the file system directories, the Java run-time system needs to know where to look for the package that we create. There are two possible ways to locate a package. First, the Java run-time system, by default, uses the current working directory. Second, a directory path or paths can be specified by setting the **CLASSPATH** environmental variable.
- If a Java package contains multiple classes, only one of them can be declared as `public` and the source file is saved with the name of the `public` class having a `.java` extension. When a source file having multiple class definitions is compiled, the compiler creates a separate `.class` file for each class.
- Both the API packages and user-defined packages can be accessed using the `import` statement. Once the required package is imported in the program, the classes belonging to that package can be used.
- We can hide the class in a package by not declaring it `public` to prevent it from being accessed outside the package.
- Java provides an approach known as interface as a convenient alternative to implement multiple inheritance.

- An interface is just like a class. The difference between an interface and a class is that an interface contains only final variables and method declarations.
- An interface is defined just like a class but rather than using the keyword `class`, the keyword `interface` is used. One or more classes can implement the interface by using the keyword `implements` in the class definition.
- All the methods must be implemented by the class which implements the interface. If a class that implements the interface does not provide complete implementation of the methods declared in the interface, then it is necessary for the class to be declared as `abstract`.
- An interface can inherit another interface by using the `extends` keyword in the same way as a class inherits from another class.
- An interface can be used to declare a set of constants that can be used by the classes implementing the interface.

NOTES

9.12 KEY WORDS

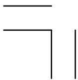
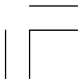
- **Package:** A named collection of classes.
- **API (Application Programming Interface) packages:** The standard packages available in Java which contain all of the standard classes.
- **User-defined packages:** Packages that are created by the users.
- **Inheritance**:The mechanism used to define a new class in terms of one or more existing classes.

9.13 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What are the advantages of using a package?
2. What are the steps for creating a package? How do you access a particular package in your file?
3. Consider a program to import a package `person` with all its classes. Identify and correct the error, if any.

```
import  person;
class ImportDemo
{
 public static void main(String[] args)
 {
    FirstClass obj=new FirstClass();
```

```
  }
}
```

NOTES

4. What do you mean by partial implementation? Explain with example.
5. State the difference between an interface and an abstract class.
6. Define interface. What are the differences and the similarities between a class and an interface?

Long-Answer Questions

1. Which of the following will generate an error and why?

```
(a) Interface1  extends  Interface2
    {
    :
    }
(b) Class1  implements  Interface1
    {
    :
    }
(c) Interface1  extends  Class1
    {
    :
    }
(d) Class1 extends Class2 implements Interface1
    {
    :
    }
```

2. Consider the following code.

```
interface InterfaceName
{
 int i=10;
 void display();
}
class ClassName implements InterfaceName
{
 void display()
 {
    System.out.println(" The value of i is: " +i);
```

NOTES

```
 }
}
class Demo
{
public static void main(String[] args)

{
    ClassName obj=new ClassName();
    obj.display();
   }
}
```

Will the code compile successfully? Justify your answer.

3. Correct the code to remove the compile-time error.

```
interface InterfaceName
{
 int a=30;
 void result();
}
class Class1 implements InterfaceName
{
public void result()
{
    System.out.println("The value of a is: "+a);
 }
}
class Class2 implements InterfaceName
{
public void show()
{
    System.out.println("This is Class2");
 }
}
public class SimpleInterface
{
 public static void main(String args[])
{
 Class1 obj1=new Class1();
 Class2 obj2=new Class2();
 obj1.result();
```

```
  obj2.show();
 }
 }
```

NOTES

4. 'Java supports the concept of multiple inheritance through interface'. Explain in detail by giving an example.
5. Consider two packages `p1` and `p2`. The variable `i` of package `p1` is accessed in package `p2`. Will this program compile successfully? Justify your answer.

```
package p1;
public class A
{
protected int i=30;
}
class B extends A
{
System.out.println("The value of i is " +i);
   }
  package p2;
  public class C extends p1.A
   {
System.out.println("The value of i is " +(i+1));
   }
```

6. Explain the steps for creating multiple public classes in a single package?

9.14 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

BLOCK - IV
MULTITHEREADING, EXCEPTION AND APPLETS

NOTES

UNIT 10 MULTITHREADED PROGRAMMING

Structure

10.0 INTRODUCTION

Most of the modern operating systems support multitasking—the ability to execute multiple programs simultaneously. Java's multithreading is a specialized form of multitasking. Multithreading is a programming concept in which a program (process) is divided into two or more subprograms (subprocesses), each of which can perform different tasks concurrently. For example, one subprogram can format the text in the text editor while another subprogram is printing the same.

In this unit, you will study about multithreaded programming, creating threads, extending the thread class, stopping and blocking a thread, life cycle of a thread, using thread methods, thread exceptions, priority, synchronization and implementing the 'runnable' interface.

10.1 OBJECTIVES

After going through this unit, you will be able to:

NOTES

- Understand the concept of threads
- Learn about the process of creating threads
- Analyse the life cycle of a thread
- Know about thread methods, exceptions and priority
- Comprehend the synchronization of threads
- Learn about inter-thread communication
- Know about suspending, resuming and stopping threads

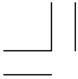
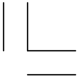
10.2 CONCEPT OF THREADS

The programs that you have learned about so far contain only a single sequential flow of control, that is, the program simply starts, performs a series of operations and ends. There is only one statement under execution at any given point of time. A thread is just like a program which has a single flow of control. It also has a starting point, an execution part and an end. Java also allows you to use multiple flows of control in a program. Such a program is known as a multithreaded program. In a multithreaded program, each thread is a separate tiny module which runs parallely with other threads. Running parallely does not mean that they are running at the same time. Most of the time, the threads run on the single processor, and only one thread is executed at a given time. However, the switching from one thread to another occurs so fast that it gives an illusion to the user that all the threads are being executed at the same time. Threads are called lightweight processes because all the threads in the main application program share the same address space in the memory.

There are several advantages of using multithreaded programs over single-threaded programs. In a traditional single-threaded environment, the CPU sits idle most of the time, as the program has to wait for each task to be completed before proceeding to the next task. In a multithreaded environment, since different tasks can be assigned to different threads, the program makes maximum utilization of the CPU keeping its idle time to a minimum. For example, one thread can read data, another can process it and a third thread can write it, thus improving the overall performance. Multithreading is best-suited for those applications that require multiple tasks to be done simultaneously.

10.2.1 Main Thread

Java programs always contain at least one thread even if you do not create any threads. This thread is called the main thread, which immediately starts executing

NOTES

when you start a program. The main thread can be used to create and start other child threads and it must be the last thread to finish execution because it performs various other actions such as shutdown and releasing resources which are used by the program. The main thread is created automatically but it can be controlled through a `Thread` object. For this, its reference is needed which can be obtained by calling the method `currentThread()`, which is a public static member of the `Thread` class. This method returns a reference to the thread on which it is called. You can control the main thread just like any other thread, once you have a reference to it.

10.3 CREATING THREADS

In Java, threads can be created in the two following ways:

- By defining a class that extends the `Thread` class
- By implementing the `Runnable` interface

In both the approaches, threads are implemented in the form of objects which contain a method called `run()`. It is the most important method in the `Thread` class. It is the entry point of a new thread and it is the place where the task to be performed by the thread is defined. The execution of a thread starts with the call to the `run()` method. The `run()` method is automatically invoked when you invoke another method of the `Thread` class called `start()`.

10.3.1 Extending Threads

You can create a thread by creating a new class that extends the `Thread` class defined in `java.lang` package and creating an instance of the class. The extending class must override the `run()` method. Inside the `run()` method, the code that needs to be executed by the thread is defined.

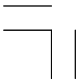
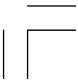
The code segment to extend the `Thread` class and override the `run()` method is

```
class ThreadName extends Thread
{
    public void run()
    {
       :  // code for the new thread
    }
}
```

Now the instance of the class `ThreadName` can be created and run using the following statements:

```
ThreadName objectname=new ThreadName();
objectname.start();
```

The second statement invokes the `start()` method after which the thread will be ready to run. It will start running once the Java runtime invokes the `run()` method.

Program 1: A program to demonstrate creating threads by extending the `Thread` class

```
class Thread1 extends Thread
{
    public void run()          //entry point of Thread1
    {
          int i=0;
          while(i<5)
       {
          System.out.println("First Child Thread:"+i);
          i=i+1;
       }
       System.out.println("\t First child exited");
    }
}
class Thread2 extends Thread
{
    public void run()      //entry point of Thread2
    {
       int j=0;
       while(j<5)
       {
          System.out.println("Second Child Thread:"+j);
          j=j+1;
       }
       System.out.println("\t Second child exited");
    }
}

class ExtendingThread
{
    public static void main(String[] args)
    {
       Thread1 firstthread=new Thread1();
       firstthread.start();    // starts the first thread
       Thread2 secondthread=new Thread2();
       secondthread.start();   // starts the second thread

       int k=0;
       while(k<5)
       {
          System.out.println("Main Thread:"+k);
          k=k+1;
       }
          System.out.println("\t Main thread exiting");
     }
}
```

Output of the program:

```
First Child Thread:0
Main Thread:0
Second Child Thread:0
```

NOTES

NOTES

```
First Child Thread:1
Main Thread:1
Second Child Thread:1
First Child Thread:2
Main Thread:2
Second Child Thread:2
First Child Thread:3
Main Thread:3
Second Child Thread:3
First Child Thread:4
Main Thread:4
Second Child Thread:4
First child exited
Main thread exiting
Second child exited
```

When you start the program, the main thread immediately starts running. The main thread then starts two child threads `Thread1` and `Thread2`, both of which perform different tasks. Once the main thread reaches the end of the `main()` method, there will be altogether three threads running concurrently on their own, in the program, `Thread1`, `Thread2` and the main thread. These three threads will run independently whenever the CPU is available for them. There is no specific order of their execution. Hence, the program may generate a different output every time you run it.

10.3.2 Implementing Runnable Interface

Another way of creating a thread is to create a class that implements the `Runnable` interface. The `Runnable` interface consists of a single method `run()`, which is required for implementing a thread. You will create a thread and pass the object of the class that implements the `Runnable` interface as an argument of the `Thread` class's constructor. The thread will now be activated by calling the `start()` method. Implementing `Runnable` is much more convenient than extending a `Thread` class when a program needs to inherit from a class apart from the `Thread` class since Java allows only a single base class.

The code segment to implement `Runnable` interface is as follows:

```
class MyNewThread implements Runnable
{
public void run()
{
      :  // code for the new thread
}
}
```

Program 2: A program to demonstrate creating threads by implementing `Runnable` interface

NOTES

```
class MyNewThread implements Runnable   // implements
Runnable
{
   public void run()               // implements run()
   method
   {
      int i=0;
      while(i<=4)
      {
         System.out.println("Child Thread: "+i);
         i++;
      }
   }
}
class RunnableInterface
{
public static void main(String[] args)
   {
      /*an object of class implementing Runnable
   interface*/
      MyNewThread runnableobj=new MyNewThread();

      /*an object of Thread class taking runnable object
      as argument*/
      Thread threadobj=new Thread(runnableobj);
      threadobj.start();
      int j=0;
      while(j<=4)
      {
         System.out.println("Main Thread: "+j);
         j++;
      }
      System.out.println("Main Thread Exiting");
   }
}
```

Output of the program:

```
Main Thread: 0
Child Thread: 0
Main Thread: 1
Child Thread: 1
Main Thread: 2
Child Thread: 2
Main Thread: 3
Child Thread: 3
Main Thread: 4
```

```
Child Thread: 4
Main Thread Exiting
```

In this example, `MyNewThread` is a class which implements the `Runnable` interface. Inside the `main()` method, an instance `runnableobj` of `MyNewThread` is created which is passed as an argument to the `Thread` class's constructor. When the new thread starts, the `run()` method of `runnableobj` is called.

NOTES

Check Your Progress

1. Define a thread.
2. Why are threads known as lightweight processes?
3. What is the main thread used for?
4. What does the `Runnable` interface consist of?

10.4 LIFE CYCLE OF A THREAD

A Java thread enters various states during its life cycle. They are as follows:

1. Newborn
2. Runnable
3. Running
4. Blocked
5. Dead

A thread is always in one of these five states. The thread can transit from one state to another as shown in Figure 10.1.

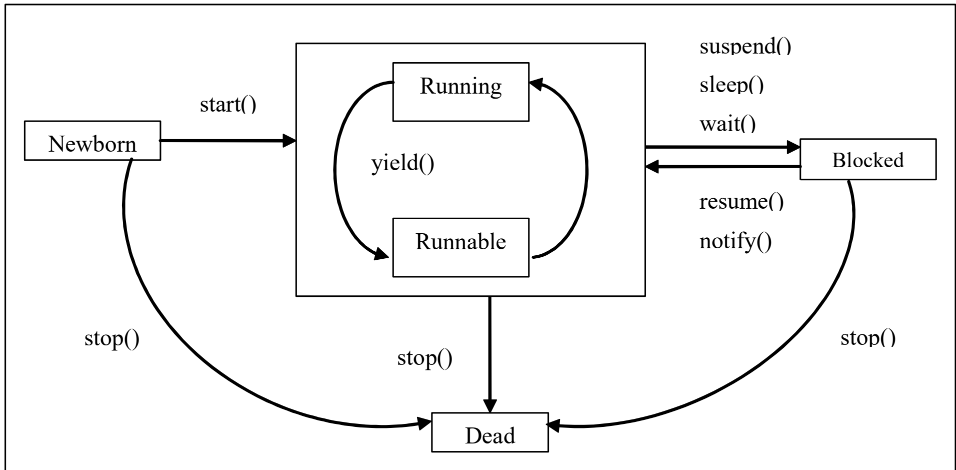


Fig. 10.1 *Life Cycle of a Thread*

NOTES

1. **Newborn state:** When a thread is created, it enters a state called the newborn state. It is the first state in the life cycle of a thread in which the thread is not yet ready to run. During this stage, you can either schedule the thread for running by calling the `start()` method or kill it by using the `stop()` method. We cannot use any other method in the newborn state as it will throw exception.

2. **Runnable state:** The thread enters the runnable state when it is ready to run and is waiting for the CPU to be allocated to it. Whenever a thread is ready to run, it is placed in the queue of threads that are waiting for the availability of the CPU. The thread scheduler picks one of the threads in the queue based on their priority. If all the threads in the queue have equal priority, the scheduler gives time slots for execution in a round–robin fashion. A thread can voluntarily give up its turn and give the control to another thread of equal priority using a static method called `yield()`. It then joins the queue at the end and waits for its turn to come.

3. **Running state:** A thread is said to be in the running state when it gets the processor for execution, that is, when the thread scheduler picks it to be the currently executing thread. The thread will be executed by the processor and will keep running until it gets preempted by a thread of higher priority or it gives up the control in one of the following conditions:

(i) ***When being suspended using the*** `suspend()` ***method:*** If you want to suspend a thread for sometime but do not want to kill it, then you can use the `suspend()` method. The suspended thread can be brought back to the execution state by invoking a method called `resume()`.

(ii) ***When it is made to sleep using the*** `sleep()` ***method:*** The `sleep()` method makes the thread sleep for a specific period of time. The thread will automatically return back to its execution state once the specified time has elapsed.

(iii) ***When it is made to wait using the*** `wait()` ***method:*** A thread can be made to wait until some events occur by calling the `wait()` method. You can make the thread run again by using the `notify()` method.

4. **Blocked state:** A thread is in a blocked state when the thread is either suspended, sleeping or waiting. The thread in this state is not considered dead, so it can re-enter the runnable state and subsequently run if the CPU is available.

5. **Dead state:** A thread is said to be in the dead state if its `run()` method is completed and it has been killed deliberately by invoking the `stop()` method at any state. Once a thread enters the dead state, it cannot re-enter any other state even if the `start()` method is invoked.

10.5 THREAD METHODS

You have seen the usage of the `start()` and `run()` method. There are various other methods of the `Thread` class. They are `yield()`, `sleep()`, `stop()`, `isAlive()` and `join()` method.

10.5.1 Using yield(), sleep() and stop() Methods

NOTES

The yield(), sleep() and stop() methods are responsible for controlling the behaviour of a thread and the transition of thread from one state to another.

The general form of sleep() method is:

```
static void sleep(long milliseconds) throws Interrupted
Exception
```

where, milliseconds is the number of milliseconds to suspend. As the sleep() method throws an exception, the call to sleep() method must be enclosed in a try block followed by a catch block otherwise, the program will not compile.

The general form of stop() method is:

```
static void stop()
```

The general form of yield() method is:

```
static void yield()
```

Program 3: A program to demonstrate the use of yield(), sleep() and stop() method

```
class Thread1 extends Thread
{
public void run()
    {
       int i=0;
       while(i<5)
       {
          if(i==2)
          yield();      //calls yield() method on Thread1
          System.out.println("First Child Thread:"+i);
          i=i+1;
       }
       System.out.println("\t First child exited");
    }
}
class Thread2 extends Thread
{
    public void run()
    {
       int j=0;
       while(j<5)
       {
```

NOTES

```
            System.out.println("Second Child Thread:"+j);
            if(j==3)
            stop();      //calls stop() method on Thread2
            j=j+1;
        }
        System.out.println("\t Second child exited");
    }
}
class Thread3 extends Thread
{
    public void run()
    {
        int k=0;
        while(k<5)
        {
            System.out.println("Third Child Thread:"+k);
            if(k==3)
            try
            {
                sleep(1000); /*calls sleep() method on
                        Thread3*/
            }
            catch(Exception e)
            {
            }
            k=k+1;
        }
        System.out.println("\t Third child exited");
    }
}
class ThreadMethods
{
    public static void main(String[] args)
    {
        Thread1 th1=new Thread1();
        Thread2 th2=new Thread2();
        Thread3 th3=new Thread3();
        th1.start();
        th2.start();
        th3.start();
```

NOTES

```
        int k=0;
        while(k<5)
        {
           System.out.println("Main Thread:"+k);
           try
           {
                 Thread.sleep(1000);
           }
           catch(Exception e)
           {
              System.out.println("Main Thread
           interrupted");
           }
           k=k+1;
        }
        System.out.println("\t Main thread exiting");

     }
  }
```

Output of the program:

```
First Child Thread:0
First Child Thread:1
Main Thread:0
First Child Thread:2
First Child Thread:3
First Child Thread:4
     First child exited
Second Child Thread:0
Second Child Thread:1
Second Child Thread:2
Second Child Thread:3
Third Child Thread:0
Third Child Thread:1
Third Child Thread:2
Third Child Thread:3
Main Thread:1
Third Child Thread:4
     Third child exited
Main Thread:2
Main Thread:3
```

```
Main Thread:4
     Main thread exiting
```

NOTES

In this example, although Thread1 started first, it relinquishes its control after implementing the while loop only two times, as the yield() method is invoked. Thread2 starts running and it is killed by invoking the stop() method. Thread3 while running, goes to sleep for 1000 milliseconds on the invocation of sleep() method.

10.5.2 Using isAlive() and join() Methods

As stated earlier, the main thread must often be the last thread to terminate since it performs several other tasks. For this, you can invoke the sleep() method within the main() method. However, it is not considered to be a good approach as the main thread has no information regarding the time when a child thread will finish its execution. In this case, the main thread has to sleep for a period long enough to assure the termination of child threads prior to its termination. For example, if the main thread is made to sleep for 2500 milliseconds and the child threads altogether take 1000 milliseconds to terminate, then there will be a delay of 1500 milliseconds for the main thread to terminate. Thus, it is necessary for the main thread to know when the child threads have finished. There are two ways to determine whether a thread has ended:

First, you can call the isAlive() method. The general form of isAlive() method is

```
final boolean isAlive()
```

This method is used to test whether a thread is still alive. It returns true if the thread is running, otherwise it returns false. The isAlive() method is used occasionally.

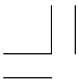
The method which is more commonly used to wait for a thread to finish is the join() method. The general form of join() method is

```
final void join() throws InterruptedException
```

This method is used to halt the execution of current thread (calling thread) until the thread on which the method is called terminates. It ensures that the current thread will wait until the called thread finishes its execution. There is another form of join() method which takes the parameter in milliseconds, which is the maximum amount of time that a thread can wait for the specified thread to terminate. As the join() method throws an exception, it must be enclosed in a try block followed by a catch block.

1.6 THREAD EXCEPTIONS

The sleep() method in a Java program is always enclosed in the try block that is followed by a catch block. The try-catch block handles the exception




that the `sleep()` method `throws`. The program does not compile, if you do not catch the exception.

NOTES

The Java run environment throws `illegalThreadStateException`, whenever you invoke a method that a thread cannot handle in its present state. For example, a sleeping thread cannot work with the `resume()` method, because a sleeping thread cannot receive any instructions. An appropriate exception handler is used to catch the exceptions that a Java method `throws`. The syntax for the `try` block is:

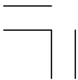
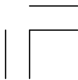
```
try
{
sleep(t)
}
```

The `try` statement is followed by the appropriate `catch` block. The four types of `catch` statement are:

- **Type 1**:
```
catch (ThreadDeath e)
{
// Kill the thread
}
```
- **Type 2**:
```
catch (InterruptedException e)
{
// Cannot handle it in this current state
}
```
- **Type 3**:
```
catch (IllegalArgumentException e)
{
// Illegal method arguments
}
```
- **Type 4**:
```
catch (Exception e)
{
// Any others exceptions
}
```

Check Your Progress

5. List the parameters to handle thread exceptions using catch statements.
6. When does the thread enter the runnable state?
7. State the various methods of the `Thread` class.

10.7 THREAD PRIORITY

NOTES

The threads you have seen so far are of equal priority in which the Java scheduler selects the thread for execution by the first-come-first-serve basis. However, each thread can be assigned for a different priority which will decide the order in which it is scheduled for running. Priorities are the integers which specify the relative priority of one thread to another. When a thread is created, it inherits its priority from the thread that created it. However, the priority of a thread can be changed by using the `setPriority()` method of the `Thread` class.

The syntax to set the priority of a thread is

```
ThreadName.setPriority(n);
```

where, `n` is an integer value which ranges from `MIN_PRIORITY` (1) and `MAX_PRIORITY` (10). The default priority is `NORM_PRIORITY` whose value is 5. `MIN_PRIORITY`, `MAX_PRIORITY` and `NORM_PRIORITY` are the constants defined in `Thread` class.

When there are multiple threads ready to be executed, the highest priority thread is chosen and executed. Only when the high priority thread stops, yields or enters the blocked state, the low priority thread starts running. However, if any higher priority thread enters, it will preempt the currently running thread forcing it to move to the runnable state.

Program 4: A program to demonstrate the assigning of priority to a thread

```
class Thread1 extends Thread
{
public void run()          //entry point of the Thread1
   {
      int i=0;
      while(i<5)
      {
         System.out.println("First Child Thread:"+i);
         i=i+1;
      }
      System.out.println("\t First child exited");
}
}
class Thread2 extends Thread
{
   public void run()       //entry point of the Thread2
   {
      int j=0;
      while(j<5)
```

```
        {
           System.out.println("Second Child Thread:"+j);
           j=j+1;
        }
        System.out.println("\t Second child exited");
     }
}
class ThreadPriority
{
     public static void main(String[] args)
     {
        Thread1 firstthread=new Thread1();
        Thread2 secondthread=new Thread2();
        // Thread2 assigned highest priority
        secondthread.setPriority(Thread.MAX_PRIORITY);

        // Thread1 assigned lowest priority
        firstthread.setPriority(Thread.MIN_PRIORITY);

        firstthread.start();
        secondthread.start();
       System.out.println("\t Main Thread Exiting");
     }
}
```

NOTES

Output of the program:

```
Second Child Thread:0
Second Child Thread:1
Second Child Thread:2
Second Child Thread:3
Second Child Thread:4
    Second child exited
    Main Thread Exiting
First Child Thread:0
First Child Thread:1
First Child Thread:2
First Child Thread:3
First Child Thread:4
    First child exited
```

NOTES

The first child thread `Thread1` has been assigned the minimum priority and the second child thread `Thread2` has been assigned the maximum priority. So despite `Thread1` being the first on which the `start()` method is called, its output is printed in the last as it has been preempted by the higher priority thread, `Thread2`.

Note: The amount of CPU time a thread gets depends not only on its priority but also on other factors such as how an operating system implements multithreading.

Check Your Progress

8. What are the main constants defined in the `Thread` class?
9. In what cases does the low priority thread start running?

10.8 SYNCHRONIZATION OF THREADS

When multiple threads need access to a single resource, there must be a way to ensure that only one thread will use the resource at any given point of time, otherwise, it may lead to a severe problem. For example, if one thread in a program reads salary from a file and another thread tries to update it, then the program may produce an undesirable output. The solution to this problem can be achieved by using a technique known as synchronization. The objective of synchronization is to control the access to shared resources.

Synchronization uses the concept of monitor. A monitor is an object which is used as a mutually exclusive lock. That is, it can be owned by only one thread at any given point of time. A thread is said to have entered the monitor when it acquires a lock. Any other thread which attempts to acquire the lock has to wait until the first thread comes out of the monitor. There are two ways to implement synchronization:

1. Synchronizing methods
2. Synchronizing statements

10.8.1 Synchronizing Methods

You can synchronize a subset (or all) of the methods of any class by using the `synchronized` keyword. When a method is declared as synchronized, Java creates a monitor. To enter the monitor, you need to call a synchronized method. Only one of the synchronized methods in a class object can execute at any given time. Java hands over the monitor to the thread that calls the method first. As long as a thread is inside a synchronized method, other threads trying to call it (or any other synchronized method) on the same instance have to wait. Only when the currently executing thread finishes executing and exits the monitor can another waiting thread enter the monitor.

NOTES

The syntax to declare a method as synchronized is

```
synchronized data_type method_name()
{
// code for the method
}
```

To understand synchronization, let us first consider a program which is not synchronized.

Program 5: A program to demonstrate synchronized method

```
class A
{
    synchronized void display(String msg)
    {
       System.out.print("(" +msg);
       try
       {
          Thread.sleep(1000);
       }catch(InterruptedException e)
       {
          System.out.println("Interrupted");
       }
       System.out.println(")");
    }
}
class MyThread extends Thread
{
    String str;
    A obj;
    MyThread (A obj1,String s)
    {
       obj=obj1;
       str=new String (s);
    }
    public void run()
    {
       obj.display(str);
    }
}

class SynchronizedMethod
{
```

NOTES

```
    public static void main(String[] args)
    {
       A obj=new A();
       MyThread th1=new MyThread(obj,"THIS");
       MyThread th2=new MyThread(obj,"IS");
       MyThread th3=new MyThread(obj,"SYNCHRONIZATION");

       th1.start();
       th2.start();
       th3.start();
    }
}
```

Output of the program:

```
(THIS)
(IS)
(SYNCHRONIZATION)
```

Note: Once a thread is in synchronized method on an instance, no other thread can enter any other synchronized method on the same instance. However, simultaneous execution of synchronized methods is possible for two different instance of the same class.

10.8.2 Synchronizing Statements

Another way of managing the execution of the thread is to synchronize a block of code or statement as this is more powerful. Synchronizing a method does not work in all cases. For example, the class you want to access is created by someone else, which does not have synchronized methods and you do not have the access rights to modify it. In this case, the access to objects of this class can be synchronized by placing the call to the methods defined by this class inside a synchronized block. However, two block of codes synchronized on the same instance cannot execute at the same time.

The general form to synchronize a block of code is

```
synchronized(object)
{
    //statements to be synchronized
}
```

where,

`object` is a reference to the object being synchronized.

10.8.3 Deadlock

Deadlock is a situation that occurs when two or more threads are in a simultaneous wait state and each of them is waiting for the release of a resource held by one of the other waiting threads. For example, consider the following code segments.

```
Thread X:
run()
{
      synchronized(obj1)
      {
        sleep(1000);
        obj2.method2();
      }

}
```

```
Thread Y:
run()
{
      synchronized(o
      {
        sleep(1000);
        obj1.method1
      }

}
```

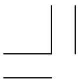
First, `Thread X` starts and synchronizes on the object `obj1` which prevents other threads to call the methods of `obj1`. `Thread X` then goes to sleep by calling the `sleep()` method and allows `Thread Y` to start. `Thread Y` starts and synchronizes on the object `obj2`. This prevents method of `obj2` to be called by any other thread. `Thread Y` goes to sleep on the invocation of `sleep()` method allowing `Thread X` to wake up. `Thread X` continues execution and tries to call `method2()` on `obj2` but it cannot call the method on `obj2` until the code in `Thread Y` that is synchronized on `obj2` finishes execution. As `Thread X` cannot proceed, `Thread Y` gets the control and tries to call `method1()` on `obj1` which is not possible until the code in `Thread X` that is synchronized on `obj1` finishes its execution. Here, neither of the threads can continue because they are deadlocked.

Program 6: A program to demonstrate deadlock

```
class A
{
    void display1( A obj2)
    {
       System.out.println("First thread waiting for second
       thread to release the resource");
       synchronized(obj2)
       {
          System.out.println("Deadlocked");
       }
    }
     void display2(A obj1)
    {
       System.out.println("Second thread waiting for first
       thread to release the resource");
       synchronized(obj1)
```

NOTES

```
        {
           System.out.println("Deadlocked");
        }
     }
}
class Thread1 extends Thread
{
     A obj1,obj2;
     Thread1(A i,A j)
     {
        obj1=i;
        obj2=j;
     }
     public void run()
     {
        synchronized(obj1)
        {
           try
           {
              sleep(1000);
           }
           catch(Exception e)
           {
              System.out.println(e);
           }
           obj2.display1(obj2);
        }
     }
}
class Thread2 extends Thread
{
     A obj1,obj2;
     Thread2(A p,A q)
     {
        obj1=p;
        obj2=q;
     }
     public void run()
     {
        synchronized(obj2)
        {
```




NOTES

```
          try
          {
             sleep(1000);
          }
          catch(Exception e)
          {
             System.out.println(e);
          }
          obj1.display2(obj1);
       }
    }
}
class Deadlock
{
    public static void main(String args[])
    {
       A obj1=new A();
       A obj2=new A();
       Thread1 t1=new Thread1(obj1,obj2);
       Thread2 t2=new Thread2(obj1,obj2);
       t1.start();
       t2.start();
    }
}
```

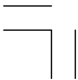
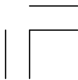
Output of the program:

```
Second thread waiting for first thread to release the
resource
First thread waiting for second thread to release the
resource
```

In this example, the thread `Thread1` owns the monitor on `obj1` and waits for the monitor on `obj2`. Similarly, the thread `Thread2` owns the monitor on `obj2` and waits for the monitor on `obj1`. `Thread1` will never release `obj1` unless it gets hold of `obj2` and `Thread2` will never release `obj2` unless it gets `obj1`. The program will never complete as the two threads are in the deadlock situation. You need to press CTRL-C to end the program.

Check Your Progress

10. What is the objective of synchronization?
11. How can subsets of methods be synchronized?

10.9 INTER-THREAD COMMUNICATION

NOTES

Inter-thread communication is a process in which multiple threads exchange messages with one another. A thread exchanges the message before or after it changes its state. There are several situations where communication between threads is important. For example, let us consider two threads A and B in which thread B uses data produced by thread A. After thread A produces data, it will keep checking every now and then whether thread B has finished using the data so that it can generate more, thus wasting CPU cycles. Thread B would again waste many CPU cycles while it waits for thread A to produce data. If threads A and B communicate with one another when they have finished their tasks, then they do not have to wait and check each other's status every time. This way, CPU cycles are not wasted.

Inter-thread communication can be achieved by using three methods, namely, `wait()`, `notify()` and `notifyall()`. These methods can be invoked only from within synchronized method or a synchronized block of code, otherwise an exception of type `IllegalMonitorStateException` is thrown. All these methods are declared `final`.

- `wait()`: It informs the current thread to release the monitor and to go into sleep state until another thread wakes it up by calling the `notify()` method. This method throws `InterruptedException`. There is another form of `wait()` method which allows us to specify the amount of time a thread can wait.
- `notify()`: It starts the first thread that called the method `wait()` on the same object.
- `notifyall()`: It starts all the threads that called the method `wait()` on the same object. In this case, the thread with the highest priority will run first.

The following is an example where a thread uses data produced by another thread without the use of `wait()` and `notify()` methods.

Program 7: A program without using `wait()` and `notify()` method

```
class P
{
    int i;
    synchronized int recieve()
    {
        System.out.println("Recieved:" +i);
        return i;
    }
    synchronized void deliver(int i)
    {
```

```
        this.i=i;
        System.out.println("Delivered:" +i);
    }
}
class Thread1 extends Thread
{
    P obj;
    Thread1(P obj1)
    {
    obj=obj1;
    }
    public void run()
    {
        int j=0;
        while(true)
        {
obj.deliver(j++);
        }
    }
}
class Thread2 extends Thread
{
    P obj;
    Thread2(P obj1)
    {
        obj=obj1;
    }
    public void run()
    {
        while(true)
        {
obj.recieve();
        }
    }
}
class NoCommunication
{
   public static void main(String[] args)
    {
        P obj=new P();
```

NOTES

NOTES

```
        Thread1 t1=new Thread1(obj);
        Thread2 t2=new Thread2(obj);
        t1.start();
        t2.start();
    }
}
```

Output of the program:

```
Delivered:0
Delivered:1
Delivered:2
Delivered:3
Delivered:4
Delivered:5
Delivered:6
Recieved:6
Recieved:6
Recieved:6
Recieved:6
Recieved:6
Delivered:7
Delivered:8
Delivered:9
```

In this example, data produced by `Thread1` is used by `Thread2`. Note that `Thread1` produces data 0 to 6 without letting `Thread2` to use it. Again, `Thread2` uses data 6, five times in a row. There is no way for the two threads to communicate with each other.

10.10 SUSPENDING, RESUMING AND STOPPING THREADS

As mentioned earlier, the predefined methods `suspend()`, `resume()` and `stop()` have been deprecated in Java2 though they are a convenient way for managing the execution of threads. These methods were deprecated as they may cause deadlocks and serious system failures in a multithreaded environment. However, in the new version of Java, suspending, resuming and stopping a thread can be performed using `boolean` type flags. The `run()` method of a thread will work based on the current flag values that indicate the execution state of a thread. For example, if the running flag is set to `true`, then the `run()` method must let the thread execute. For the `run()` method to suspend the execution of

the currently running thread, the suspend flag must be set to `true`. Likewise, the thread will die once the stop flag is set to `true`.

NOTES

Program 8: A program to demonstrate suspend, resume and stop operations

```
class ChildThread extends Thread
{
    boolean suspend_flag,stop_flag;
    String name;
    ChildThread(String str)
    {
    name=str;
       suspend_flag=false;
       stop_flag=false;
    }
    public void run()
    {
       try
       {
          int i=5;
          while(i>=1)
          {
             System.out.println(name+" "+i);
             sleep(1000);
             i—;
             synchronized(this)
             {
                while(suspend_flag)
                {
                 wait();
                 if(stop_flag)
                 {
                      break;
                 }
                }
             }
          }
       }catch(InterruptedException e)
       {
          System.out.println("Thread interrupted");
       }
```

NOTES

```
    }
    synchronized void my_suspend()
    {
       suspend_flag=true;
    }
    synchronized void my_resume()
    {
       suspend_flag=false;
       notify();
    }
    synchronized void my_stop()
    {
       suspend_flag=false;
       stop_flag=true;
       notify();
    }
}
class SRS
{
public static void main(String[] args)
{
       try
    {
       ChildThread obj=new ChildThread("Thread");
       obj.start();
       System.out.println("Thread started");
       Thread.sleep(2000);
       obj.my_suspend();
       System.out.println("Thread is suspended");
       Thread.sleep(2000);
       obj.my_resume();
       System.out.println("Thread is resumed");
       Thread.sleep(2000);
       obj.my_suspend();
       System.out.println("Thread is suspended");
       Thread.sleep(2000);
       obj.my_resume();
       System.out.println("Thread is resumed");
       Thread.sleep(2000);
       obj.my_stop();
       System.out.println("Thread stopped");
```

NOTES

```
    }
    catch(InterruptedException e)
    {
       System.out.println("Thread interrupted");
    }

  }
}
```

Output of the program:

```
Thread started
Thread 5
Thread 4
Thread 3
Thread is suspended
Thread is resumed
Thread 2
Thread 1
Thread is suspended
Thread is resumed
Thread stopped
```

Check Your Progress

12. What is inter-thread communication?
13. How can inter-thread communication be achieved?
14. How is the suspension, resuming and stopping of thread performed in the new version of Java?

10.11 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

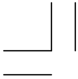
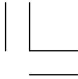
1. A thread is a program which has a single flow of control.
2. Threads are called lightweight processes because all the threads in a main application program share the same address space in the memory.
3. The main thread is used to create and start other child threads and it must often be the last thread to finish execution because it performs various other actions.
4. The `Runnable` interface consists of a single method `run()`, which is required for implementing a thread.

NOTES

5. Inter-thread communication is a process in which multiple threads exchange messages with one another.
6. The thread enters the runnable state when it is ready to run and is waiting for the CPU to be allocated to it.
7. The various methods of the `Thread` class are `yield()`, `sleep()`, `stop()`, `isAlive()` and `join()` method.
8. `MIN_PRIORITY`, `MAX_PRIORITY` and `NORM_PRIORITY` are the constants defined in the `Thread` class.
9. The low priority thread starts running only when the high priority thread stops, yields or enters the blocked state.
10. The objective of synchronization is to control the access to shared resources.
11. A subset (or all) of the methods of any class can be synchronized by using the `synchronized` keyword.
12. Inter-thread communication is a process in which multiple threads exchange messages with one another.
13. Inter-thread communication can be achieved by using three methods, namely, `wait()`, `notify()` and `notifyall()`.
14. In the new version of Java, suspending, resuming and stopping a thread can be performed using `boolean` type flags.

10.12 SUMMARY

- Multithreading is a programming concept in which a program (process) is divided into two or more subprograms (subprocesses), each of which can perform a different task concurrently.
- A thread is just like a program which has a single flow of control. It also has a starting point, an execution part and an end.
- Threads are called lightweight processes because all the threads in a main application program share the same address space in the memory.
- Java programs always contain at least one thread even if you do not create one. This thread is called the main thread and it is the one which immediately starts executing when you start a program.
- In Java, threads can be created in two ways, by extending the `Thread` class and by implementing the `Runnable` interface. In both the approaches, threads are implemented in the form of objects which contain a method called `run()`. It is the entry point of a new thread and it is the place where the task to be performed by the thread is defined.
- There are various states that a Java thread can enter during its life cycle. They are newborn, runnable, running, blocked and dead state. A thread is always in one of these five states.

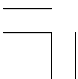
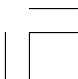



NOTES

- The `yield()`, `sleep()` and `stop()` methods are responsible for controlling the behaviour of a thread and the transition of thread from one state to another.
- There are two ways to determine whether a thread has ended—the `isAlive()` method and the `join()` method.
- Priorities are the integers which specify the relative priority of threads. When a thread is created, it inherits its priority from the thread that created it. The priority of a thread can be changed by using the `setPriority()` method of the `Thread` class.
- When multiple threads need access to a single resource, there is a technique called synchronization to ensure that only one thread will use the resource at any given point of time.
- There are two ways to implement synchronization—one is to use the synchronize method and another is to synchronize statements. The keyword `synchronized` is used in both the cases.
- Deadlock is a situation that occurs when two or more threads are in a simultaneous wait state and each of them is waiting for the release of a resource held by one of the other waiting thread.
- Inter-thread communication is a process in which multiple threads exchange messages with one another. A thread exchanges message before or after it changes its state.
- Inter-thread communication can be achieved by using three methods, namely, `wait()`, `notify()` and `notifyall()`.
- The predefined methods `suspend()`, `resume()` and `stop()` have been deprecated in Java2 as they may cause deadlocks and serious system failures in a multithreaded environment.
- In the new version of Java, suspending, resuming and stopping a thread can be performed using `boolean` type flags.

10.13 KEY WORDS

- **Multitasking:** It refers to the concurrent execution of two or more tasks by the CPU of a computer.
- **Deadlock:** It is a situation that occurs when two or more threads are concurrently in a wait state.
- **Synchronization:** It is a method that allows the execution of methods or threads in a specific format—either statement by statement or by blocks of statements.

10.14 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. What is multithreading?
2. How do you set priorities for threads?
3. What is synchronization?
4. When are two threads said to be deadlocked?
5. What are the general forms of the `isAlive()` and `Join()` methods?
6. Define the concept of `Thread` exception
7. Enumerate the advantages of synchronizing statements.
8. What are situations in which inter-thread communication occur?

Long-Answer Questions

1. Explain the concept of thread in Java. What is the advantage of using a multithreaded program over a single-threaded program?
2. What are the two ways of creating a thread in Java? Give an example of each.
3. Describe the complete life cycle of a thread.
4. Write a program to illustrate the use of `stop()` and `suspend()` methods.
5. Explain the different ways to implement synchronization by the help of examples.
6. Write the correct code that demonstrates the `sleep()` method.
7. Write a program which implements `Runnable` interface.

10.15 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

NOTES

UNIT 11 MANAGING ERROR AND EXCEPTIONS

Structure

11.0 INTRODUCTION

An exception (or exceptional event) is a problem that arises during the execution of a program. When an 'Exception' occurs the normal flow of the program is disrupted and the Program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. An exception can occur for many different reasons. The exception occurs when a user has entered an invalid data, a file that needs to be opened cannot be found, a network connection has been lost in the middle of communications or the JVM has run out of memory, etc.

An exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e., at run time, that disrupts the normal flow of the program's instructions. An 'Error' indicates serious problem that a reasonable application should not try to catch while the 'Exception' indicates conditions that a reasonable application might try to catch.

In this unit, you will study about the types of errors, exceptions, syntax of exception handling code, multiple catch statements, using finally statement, throwing our own exceptions, graphics programming: the graphics class, lines and rectangles, circles and ellipses, drawing arcs, drawing polygons and line graphs.

In this unit, you will also learn about exception handling in Java. Exception is an unpredicted event that occurs while the program is executing, and thus disrupts the normal flow of the program or terminates the program abnormally. Therefore, exception handling is used to provide solution to these exceptions, if a solution is not possible then to provide an error message alerting the user.

After reading this unit, you will be able to work with Java output presentation using `Graphics` class, `Font` class and `Image` class.

11.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the concept of managing errors
- Identify the types of errors in Java programs
- Know the importance of exception handling
- Create a user defined exception
- Describe various methods and classes available in Graphics class
- Learn about the Java output presentation using Graphics class, Font class and Image class

11.2 TYPES OF ERRORS

Error is an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing.

The most common errors can be broadly classified as follows:

1. **Run Time Error:** Run Time errors occur or we can say, are detected during the execution of the program. Sometimes these are discovered when the user enters an invalid data or data which is not relevant. Runtime errors occur when a program does not contain any syntax errors but asks the computer to do something that the computer is unable to reliably do. During compilation, the compiler has no technique to detect these kinds of errors. It is the JVM (Java Virtual Machine) which detects it while the program is running. To handle the error during the run time we can put our error code inside the try block and catch the error inside the catch block.
2. **Compile Time Error:** Compile Time Errors are those errors which prevent the code from running because of an incorrect syntax such as a missing

semicolon at the end of a statement or a missing bracket, class not found, etc. These errors are detected by the java compiler and an error message is displayed onto the screen while compiling. Compile Time Errors are sometimes also referred to as Syntax errors. These kind of errors are easy to spot and rectify because the java compiler finds them for you. The compiler will tell you which piece of code in the program got in trouble and its best guess as to what you did wrong. Usually, the compiler indicates the exact line where the error is, or sometimes the line just before it, however, if the problem is with incorrectly nested braces, the actual error may be at the beginning of the block. In effect, syntax errors represent grammatical errors in the use of the programming language.

3. **Logical Error:** A logic error is when your program compiles and executes, but does the wrong thing or returns an incorrect result or no output when it should be returning an output. These errors are detected neither by compiler nor by JVM. The Java system has no idea what your program is supposed to do, so it provides no additional information to help you find the error. Logical errors are also called Semantic Errors. These errors are caused due to an incorrect idea or concept used by a programmer while coding. Syntax errors are grammatical errors whereas, logical errors are errors arising out of an incorrect meaning.

Check Your Progress

1. Define errors in Java.
2. Differentiate between run time error and compile time error.

11.3 EXCEPTION HANDLING

The term exception is an abbreviation for the phrase 'exceptional event'. It is an unpredicted event that occurs while the program is executing, and thus disrupts the normal flow of the program or terminates the program abnormally. The errors that cause exceptions can range from simple programming errors, such as opening or reading an invalid file or accessing an invalid index of an array to severe errors, such as running out of memory.

When an error occurs within a method, the Java runtime system creates an exceptional object. This exceptional object contains information about the exception which includes its type and the state of the program when the error occurred. Once an exceptional object is created, it is thrown and the runtime system searches for a method to handle the exception. It keeps on searching for a method that contains an appropriate exception handler. An exception handler is considered

NOTES

appropriate if the type of exception handled by the handler is same as the type of the exception thrown.

Before understanding the significance of the exception handling mechanism, it is necessary to see what happens if the exceptions are not handled.

Program 1: A program to demonstrate an unhandled exception.

```
class ExceptionExample
{
public static void main(String args[])
    {
       int a=15;
       int b=3;
       int c=a/(a-(5*b));  //exception occurs here
       System.out.println("Result: "+c);
    }
}
```

Output of the program:

```
Exception in thread "main" java.lang.ArithmeticException:
/ by zero at ExceptionExample.main(ExceptionExample.java:7)
```

In this program, no compile time error occurs. However, when the division by zero statement is executed a runtime error is generated. At this point, the Java runtime system creates an exceptional object and throws it. The thrown object expects an exception handler to handle it. Since no exception handler is provided in this program, Java's default exception handler is invoked. The default exception handler will display the error message and terminate the program abruptly.

11.3.1 Handling of Exception

The mechanism of exception handling in Java not only guarantees the detection and handling of runtime errors but also provides a way to separate the error handling code from the rest of the program. This makes the program less complex, more readable and efficient, as the normal execution path is not interrupted for checking errors. Moreover, the routine of exception handling (called exception handler) is invoked automatically whenever an error occurs.

Java exception handling is governed by the five following keywords:

- `try`
- `catch`

- `finally`
- `throw`
- `throws`

NOTES

A set of statements that needs to be monitored for the exceptions is contained in the `try` block. Whenever an exception occurs within the `try` block, it is thrown. This passes the control to the `catch` block which handles the exception appropriately. The Java runtime system automatically throws system generated exceptions. If the user wants to throw the exception explicitly, the `throw` keyword is used. Sometimes, a method may throw exceptions which it cannot handle. This must be specified by using the `throws` keyword. The code that must be executed whether an exception is thrown or not is kept within the `finally` block, which is optional.

11.3.2 Types of Exceptions

Java provides several built-in classes which define all types of exceptions. These exception classes are arranged in a hierarchy having a `Throwable` class on the top (see Figure 11.1). This means that the `Throwable` class is the superclass and all the exception classes inherit methods defined by it. Two immediate subclasses of the `Throwable` class are the `Exception` class and `Error` class.

- **`Exception` Class:** It defines those exceptions which are thrown by methods of the standard Java class library or methods defined in the user's program and can be trapped within the program. That is, the program can reasonably recover from these types of exceptions. This class is also used (inherited) when the users want to create their own exceptions in the application.
- **`Error` Class:** It defines those exceptions that do not occur frequently and are difficult to be recovered from. For example, if a class file is missing or system runs out of memory.

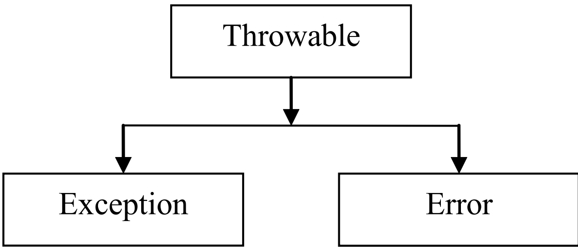


Fig. 11.1 *Java Exception Hierarchy*

NOTES

Some of the most commonly used exceptions that will be encountered are listed in Table 11.1.

Table 11.1 *Some Common Exceptions in Java*

Exception	Description
`ArithmeticException`	Thrown when arithmetic error occurs in the program, such as divide-by-zero.
`NullPointerException`	Thrown when the user tries to use an object without initializing the object or in other words when an object that has not been allocated memory is used.
`IOException`	Thrown when error occurs during input/output of data.
`ArrayIndexOutOfBoundsException`	Thrown when an attempt is made to access an array element with invalid index value.
`ArrayStoreException`	Thrown when an attempt is made to store the incompatible type of data in an array.
`IllegalAccessException`	Thrown when an illegal attempt is made to access a class.
`NumberFormatException`	Thrown when an invalid conversion of a string to a numeric format takes place.
`StringIndexOutOfBoundsException`	Thrown when an attempt is made to access a string element that is beyond the index of the string.
`IllegalArgumentException`	Thrown when an illegal argument is used to invoke a method.
`NegativeArraySizeException`	Thrown when an array of negative size is created.

11.3.3 Using `try` and `catch` Blocks

The default exception handler provided by the Java runtime system does not prevent the abrupt termination of the program. To prevent this, Java provides the facility to construct your own exception handler, by which you can fix the errors yourself. This can be achieved by enclosing the code that may throw an exception within a `try` block. The `try` block is enclosed by curly braces and preceded by the keyword `try`. Whenever, an exception occurs within the `try` block, it is thrown. This passes the control to the `catch` block associated with the `try` block.

The syntax to define `try-catch` block is as follows:

```
try //try block begins
{
    //code that may cause an exception

}   //try block ends
catch(exception_type ex)
{
    //code to handle the exception
}
```

NOTES

If the first statement of the try block causes an exception, the remaining statements are not executed and the control passes to the catch block. The catch statement requires a single argument which is of the same type as of the exception that needs to be handled. This exception type must be a subclass of Throwable class. It is not necessary that every time the program is executed, an exception occurs. If an exception is not thrown, the catch block is skipped and the control passes to the statement immediately following the catch block.

The try and catch blocks form a logical unit. The scope of the catch block is limited only to those statements which are enclosed within the immediately preceding try block.

Note: Compile time error is generated if the try statement is not followed by any catch statement.

Program 2: A program to illustrate handling of an exception using try and catch blocks.

```
class TryCatchBlock
{
    public static void main(String args[])
    {
       int a=15;
       int b=3;
       int c=0;
       try
       {
          System.out.println("try block begins");
          c=a/(a-(5*b));   //exception generated
          System.out.println("try block ends");
       }
       catch(ArithmeticException ae)
       {
          System.out.println("Arithmetic Exception is
       caught here");
          System.out.println("The resultant value of c
       is: "+c);
       }
       c=a/b;
       System.out.println("New value of c is: " +c);
    }
}
```

NOTES

Output of the program:

```
try block begins
Arithmetic Exception is caught here
The resultant value of c is: 0
New value of c is: 5
```

In this example, the exception generated within the `try` block is caught inside the `catch` block thus, preventing the abnormal termination of the program as seen in program 4.

11.3.4 Multiple `catch` Blocks

It is not necessary that the code enclosed within the `try` block throws a single exception. In case, multiple exceptions are thrown within a `try` block, Java allows the use of multiple `catch` blocks for handling all these exceptions.

The syntax to define multiple `catch` blocks is

```
try             //try block
{
//code that may cause exceptions
}
catch (exception_type e1)      //catch block 1
{
...
}
catch (exception_type e2)      //catch block 2
{
...
}
...
catch(exception_type en)       //catch block N
{
...
}
```

There is only one `try` block from which the exception is thrown and depending on the type of exception thrown, the corresponding `catch` block will be executed. Whenever, an exception is thrown, the `catch` blocks are searched in sequential order for an appropriate match. The first `catch` block, whose parameter type matches with the type of exception, gets executed and other `catch` blocks are ignored. Once the execution of the appropriate `catch` block is over, the control passes to the statement immediately following the last `catch` block.

NOTES

11.3.5 Nested `try` Blocks

The `try` blocks can be nested, that is, one `try-catch` block can be placed inside another `try-catch` block. If an exception occurs within a particular `try` block, then the `catch` blocks associated with this `try` block are searched for an appropriate match. If no match is found then the control passes to the next outer `try-catch` block. This process continues until an appropriate match is found. If no match is found, the program terminates abnormally.

The syntax of the nested `try` block is

```
try //outer try block
{
    try   //inner try block
    {
    ...
    }
    catch    //inner catch block
    {
    ...
    }
}
catch     //outer catch block
{
...
}
```

11.3.6 Using `finally` Block

It has been observed that when an exception is thrown in the program, the remaining statements in the `try` block are not executed and the control directly gets transferred to the subsequent `catch` block. However, there are certain statements in the program that need to be executed whether or not the exception is raised. For this, Java provides the `finally` keyword. The code within the `finally` block will always be executed whether or not the exception is thrown. If an exception is raised with a matching `catch` block, then the `finally` block gets executed after the execution of that `catch` block. On the other hand, if no matching `catch` block is found then also the `finally` block is executed after the execution of the `try` block. The `finally` block is optional, however, it is necessary to include either `catch` or `finally` block with `try` block.

Program 3: A program to illustrate the use of `finally` block.

```
class FinallyBlock
{
    public static void main(String args[])
    {
```

NOTES

```
        int a=67;
        int b=0;
        try
        {
           System.out.println("The value of a: " +a);
           System.out.println("The value of b: " +b);
           int c=(a/b); //exception thrown
           System.out.println("Result is: " +c);
        }
        catch(Exception e)
        {
           System.out.println(e); //prints the
        corresponding exception
        }
        finally
        {
           System.out.println("Denominator cannot be
           zero");
        }
    }
}
```

Output of the program:

```
The value of a: 67
The value of b: 0
java.lang.ArithmeticException: / by zero
Denominator cannot be zero
```

Here, the thrown exception is caught in the `catch` block and an appropriate error message is displayed. After that, the statement within the `finally` block is executed.

11.3.7 Throw Keyword

As you already know, Java runtime system automatically throws system generated exceptions. However, Java provides a mechanism to throw an exception explicitly by using the `throw` keyword.

The syntax of `throw` statement is as follows:

```
throw ExceptionObject;
```

where, `ExceptionObject` is an object of `Throwable` class or its subclass.

When a `throw` statement is encountered in the program, the execution of the subsequent statements in the `try` block stops and the corresponding `catch` block is searched. The nearest `try` block is checked to determine if it contains a

catch block to match the exception of its type. If it is found, then that catch block is executed else subsequent try blocks are inspected. In case, if no matching catch block is found, then the, default exception handler comes into action and stops the normal execution of the program and displays the error message on the output screen.

It should be noted that instances of classes other than Throwable class or its subclasses cannot be used as exception objects. The Throwable object can be created using a new operator or using a parameter inside the catch clause.

Program 4: A program to demonstrate the use of throw keyword.

```
class ThrowExampleDemo
{
    public static void main(String args[])
    {
       try
       {
          ThrowExample(); //invoking ThrowExample() method
       }
       catch(ArithmeticException ae)
       {
          System.out.println("The exception is recaught
          here: "+ae);
       }
    }
    static void ThrowExample()
    {
       int a=0;
       int b=6;
       try
       {
          int c=b/a;        //system-generated exception
          System.out.println(c);
       }
       catch(ArithmeticException aoe)
       {
          System.out.println("The exception is caught
       inside the method ThrowExample");
          throw aoe;       //exception thrown explicitly
          }
    }
}
```

NOTES

Output of the program:

```
The exception is caught inside the method ThrowExample
The exception is recaught here:java.lang.
ArithmeticException: / by zero
```

In this example, the `ArithmeticException` occurs inside the `ThrowExample()`. This exception is caught inside the `catch` block inside the same method, which explicitly rethrows it using the `throw` keyword. This is called rethrowing of the exception. Now the control passes back to the `catch` block of the `main()` method and the thrown exception is again caught here.

Using `throws` Keyword

Sometimes, a method may generate an exception but cannot handle it. That is, there may be a method in the program which is generating (throwing) an exception but it does not have the appropriate exception handing mechanism. The methods that call such methods must be cautioned about this behaviour so that calling methods can take appropriate measures to safeguard themselves against the exceptions. This is done by appending the `throws` keyword after the method name in the method declaration statement. The `throws` clause includes all types of exceptions excluding those belonging to `Error` or `Runtime` classes or their subclasses. All other exceptions which a method may throw must be listed after the `throws` keyword in the method declaration, otherwise compile time error is generated.

The syntax of the `throws` clause is

```
return_type method_name() throws exception_list
{
    //body of the method
}
```

where, `exception_list` includes all the exceptions that the method might throw.

11.3.8 Creating Your Own Exceptions

You have studied some of the built-in exceptions provided by the Java platform. These built-in exceptions are used to handle the errors occurring in the program. But sometimes the programmer wants to create his own application specific exceptions. User defined exceptions can be created by defining a subclass of `Exception` class and using the `throw` keyword.

Program 5: A program to demonstrate the creation of user defined exceptions.

```
class OwnException extends Exception
{
    public OwnException(String message) //Constructor
    {
```

```
      System.out.println("OwnException class
   constructor");
   }
}
public class CreateOwnException
{
      public static int  divide(int i,int j) throws
   OwnException
      {
         if(j==0)
         throw new OwnException("This statement is not
         executed");
         return i/j;
      }
      public static void main(String[] args)
      {
         int i=70;
         int j=0;
         try
         {
            System.out.println("The value of i: " +i);
            System.out.println("The value of j: " +j);
            System.out.println("Division of two numbers
            is: " +divide(i,j));
         }
         catch (OwnException oe)
         {
            System.out.println("Denominator can not be
            zero");
            oe.printStackTrace();
         }
         catch(Exception e)
         {
            System.out.println(e.getMessage());
         }
      }
   }
```

NOTES

Output of the program:

```
The value of i: 70
The value of j: 0
```

NOTES

```
OwnException class constructor
Denominator can not be zero
OwnException
at CreateOwnException.divide(CreateOwnException.java:13)
at CreateOwnException.main(CreateOwnException.java:24)
```

This example defines a subclass of `Exception` called `OwnException`, which inherits the methods defined by the `Throwable` class through `Exception` class. The `CreateOwnException` class defines a method `divide()` which throws an object `oe` of `OwnException` type when the denominator of the fraction is zero. The `printStackTrace()` method of the `Throwable` class is overridden to display the message which contains all the information that lead to the error, such as the name of the method which caused the error, the name of the caller of the method which lead to the error, the line number in which that error has occurred, and so on.

Check Your Progress

3. What do you understand by exception handling?
4. Why `finally` is used in exception handling?
5. Define `IOException`.
6. Write the syntax of `try-catch` block.
7. What do you understand by `throws` keyword in exception handling?
8. What is Java built-in class hierarchy that defines all types of exceptions?

11.4 GRAPHICS PROGRAMMING

So far, you have learned how to create and run a simple Java applet. Java also allows the creation of an applet, which uses various classes to draw figures of different shapes, add images, display text in different fonts and styles, use and create various colors, etc.

Using `Graphics` Class

Java allows graphical elements in an applet via `Graphics` object. `Graphics` object is an instance of the class `java.awt.Graphics`. Each `Graphics` object has its own coordinate system and the methods of `Graphics` class include drawing shapes, lines, rectangles, circles, polygons and more. You can access the `Graphics` object through the `paint(Graphics gra)` method of your applet where `gra` is the particular `Graphics` object with which you can draw.

NOTES

Drawing Lines

A line can be drawn using the `drawLine()` method of the `Graphics` class. This method takes four parameters, which represent the coordinates of the end points of the line.

The general form of the `drawLine()` method is

```
void drawLine(int a1, int b1, int a2, int b2)
```

where,

`a1,b1` is the coordinate of the starting points of the line and `a2,b2` is the coordinate of the starting points of the line.

For example, the statement `gra.drawLine(20,100,90,100)` will draw a straight line from the coordinate point `(20,100)` to `(90,100)` as shown in Figure 11.2.

(20, 100) **(90, 100)**

Fig. 11.2 *A Straight Line having Coordinates (20,100) and (90,100)*

Drawing and Filling Rectangles

A rectangle can be drawn by using the `drawRect()` method and it also takes the four parameters.

The general form of the `drawRect()` method is as follows:

```
void drawLine(int a1, int b1, int w, int h);
```

where, `a1,b1` is the coordinate of the top left corner of the rectangle, `w` is the width of the rectangle and `h` is the height of the rectangle.

For example, the statement `gra.drawRect(20,20,50,30)` will draw a rectangle starting at `(20,20)` with width of `50` pixels and height of `30` pixels as shown in Figure 11.3.

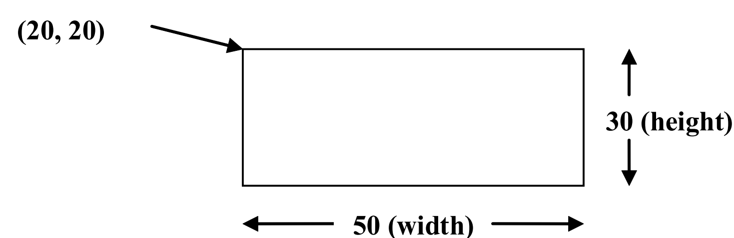


Fig. 11.3 *A Rectangle with Width 50 pixels and Height 30 pixels*

Note that the `drawRect()` method draws only the boundary of the rectangle. To draw a solid (filled) rectangle, `fillRect()` method is used. This method also takes four parameters similar to the `drawRect()` method.

NOTES

To draw a solid rectangle having aforementioned parameters, we use the statement `gra.fillRect(20,20,50,30)`, which draws the rectangle as shown in Figure 11.4.

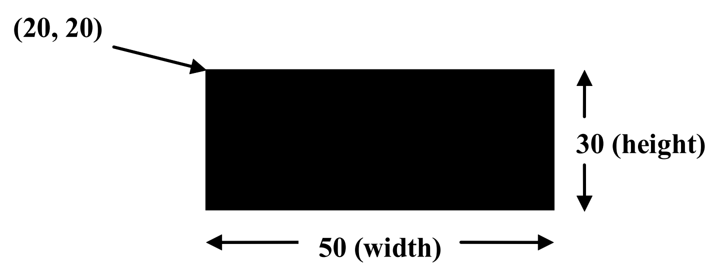


Fig. 11.4 A Filled Rectangle having Width 50 pixels and Height 30 pixels

A rounded outlined rectangle can be drawn by using the `drawRoundRect()` method. This method takes the six parameters (see Figure 11.5).

The general form of the `drawRoundRect()` method is

```
void drawRoundRect(int a1, int b1, int w, int h, int
xdia, int ydia);
```

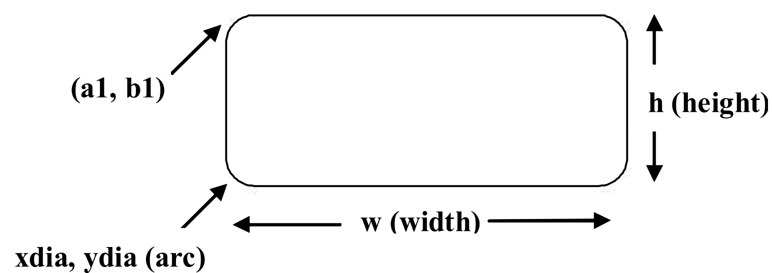


Fig. 11.5 A Rounded Rectangle

where, `xdia` is the diameter of the rounding arc (along X-axis) and `ydia` is the diameter of the rounding arc (along Y-axis).

Similarly, a rounded filled rectangle can be drawn using `drawfillRoundRect()` method. This method also takes six parameters similar to the `drawRoundRect()` method.

Note: All the shapes are drawn relative to the Java's coordinate system. The origin `(0, 0)` of the coordinate system is located at its upper-left corner such that the positive `x` values are to its right and the positive `y` values are to its bottom.

Drawing and Filling Ellipses and Circles

An ellipse can be drawn using the `drawOval()` method. The ellipse is drawn within an imaginary bounding rectangle. This method takes four arguments in which the first two represent the top left corner of the bounding rectangle and the next two represent the width and height of the oval or the bounding rectangle (see Figure 11.6).

NOTES

The general form of the `drawOval()` method is as follows:

```
void drawOval(int a1, int b1, int w, int h);
```

where, `a1, b1` is the coordinate of the top left corner of the bounding rectangle, `w` is the width of the bounding rectangle and `h` is the height of the bounding rectangle.

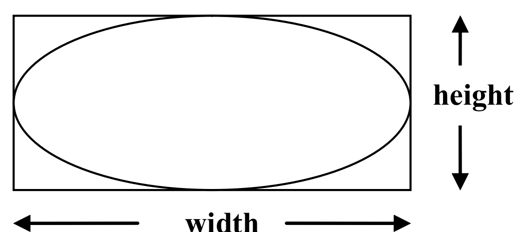


Fig. 11.6 *An Ellipse*

Similarly, a circle can be drawn using this method but the dimension of width and height should be same. That is, the bounding rectangle must be a square.

Similar to the rectangle methods, the `drawOval()` method draws the boundary of an oval and the `fillOval()` method draws a solid oval.

Drawing Arcs

An arc can be drawn using the `drawArc()` method. This method takes six arguments in which the first four are same as the arguments of the `drawoval()` method and the next two represents the starting angle of the arc and the sweep angle around the arc, respectively.

The general form of the `drawArc()` method is as follows:

```
void drawArc(int a1, int b1, int w, int h, int strt_angle,
int sweep_angle);
```

where, `a1,b1` is the coordinate of the top left corner of the bounding rectangle, `w` is the width of the bounding rectangle, `h` is the height of the bounding rectangle, `strt_angle` is the starting angle of the arc (in degrees) and `sweep_angle` is the number of degrees (angular distance) around the arc (in degrees).

The arc shown in Figure 11.7 has the starting angle as 0^0 degrees and sweep angle as 135^0.

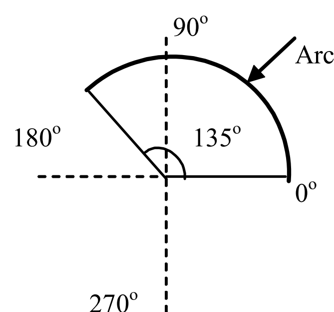


Fig. 11.7 *An Arc of 135^0 Sweep Angle*

NOTES

You can also draw filled arcs using the `fillArc()` method.

Drawing Polygons

A polygon is a closed geometrical figure, which can have any number of sides. A polygon can be drawn by using the `drawPolygon()` method. This method takes the three parameters (see Figure 11.8).

The general form of the `drawPolygon()` method is

```
void drawPolygon(int a[], int b[], int n);
```

where, `a[]` is the array of integers having x-coordinates, `b[]` is the array of integers having y-coordinates and `n` is the total number of coordinate points required to draw a polygon.

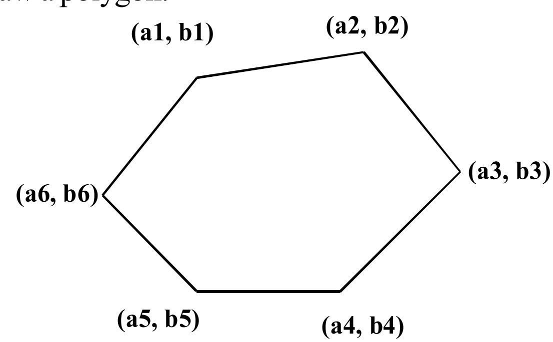


Fig. 11.8 *A Polygon having Six Sides*

Program 6: An applet code to demonstrate the use of various methods of `Graphics` class

```
import java.awt.*;
import java.applet.*;
public class GraphicsExample extends Applet
{
public void paint(Graphics gra)
    {
        gra.drawRect(10,40,80,40);
        gra.fillRect(130,40,80,40);
        gra.drawRoundRect(250,40,80,40,8,8);
        gra.fillRoundRect(370,40,80,40,8,8);

        gra.drawOval(0,125,80,40);
        gra.fillOval(120,125,80,40);

        gra.drawArc(240,125,80,40,0,180);
        gra.fillArc(370,125,80,40,0,180);

        int x[]={100,150,200,170,130,100};
```

NOTES

```
        int y[]={250,200,250,300,300,250};
        int n=x.length;
        gra.drawPolygon(x,y,n);

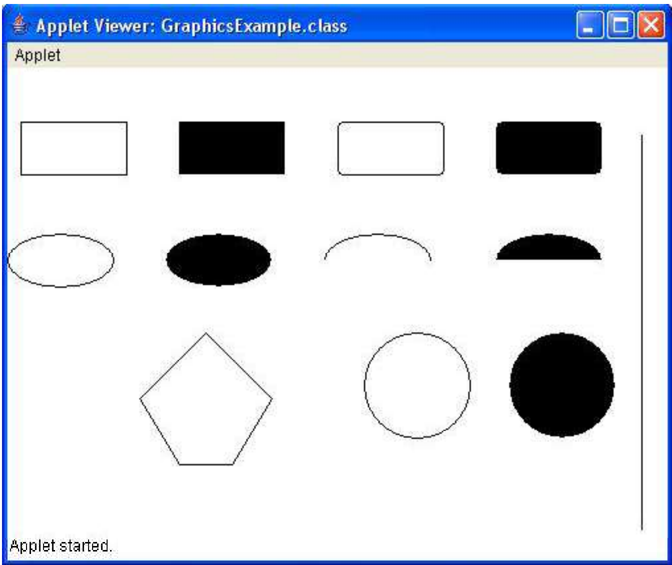
        gra.drawOval(270,200,80,80);
        gra.fillOval(380,200,80,80);

gra.drawLine(480,50,480,350);
    }
}
```

The HTML code for `GraphicsExample` is as follows:

```
<HTML>
    <HEAD>
    </HEAD>
    <BODY>
        <CENTER>
            <APPLET
                CODE="GraphicsExample.class"
                WIDTH    =600
                HEIGHT=350>
            </APPLET>
        </CENTER>
    </BODY>
</HTML>
```

Output of the program:



NOTES

Using `Color` Class

The `Color` class provides various methods to use any color you want in the display. It defines the various color constants which can be directly used only by specifying the color of your choice. In addition, the `Color` class allows creation of millions of colors. The `Color` class contains three primitive colors namely, red, blue and green, and all other colors are a combination of these three colors.

One of the constructors that is used to create color of your choice is as follows:

```
Color(int red, int green, int blue);
```

where, `red`, `green`, `blue` can take any value between `0` and `255`.

Setting Background and Foreground Color

To set the color of the background of an applet window, `setBackground()` method is used.

The general form of the `setBackground()` method is as follows:

```
void setBackground(mycolor);
```

Similarly, to set the foreground color to a specific color, that is, the color of text, `setForeground()` method is used.

The general form of the `setForeground()` method is as follows:

```
void setForeground(mycolor);
```

where, `mycolor` is one of the color constants or the new color created by the user.

The list of `color` constants is as follows:

- `Color.red`
- `Color.orange`
- `Color.gray`
- `Color.darkGray`
- `Color.lightGray`
- `Color.cyan`
- `Color.pink`
- `Color.white`
- `Color.blue`
- `Color.green`
- `Color.black`
- `Color.yellow`

Program 7: An applet code to demonstrate the use of `Color` class.

```
import java.applet.*;
import java.awt.*;
```

NOTES

```
public class ColorExample extends Applet
{
    Color c1,c2;
    public void init()
    {
       //creating new colors
       Color c1=new Color(0,0,255);
       Color c2=new Color(100,220, 190);
    }
    public void paint(Graphics gra)
    {
    setBackground(Color.white);//setting background
       color

       //drawing a line of color c1
       gra.setColor(c1);
       gra.drawLine(10,20,150,60);

       //drawing a solid oval of color c2
       gra.setColor(c2);
       gra.fillOval(10,50,100,200);

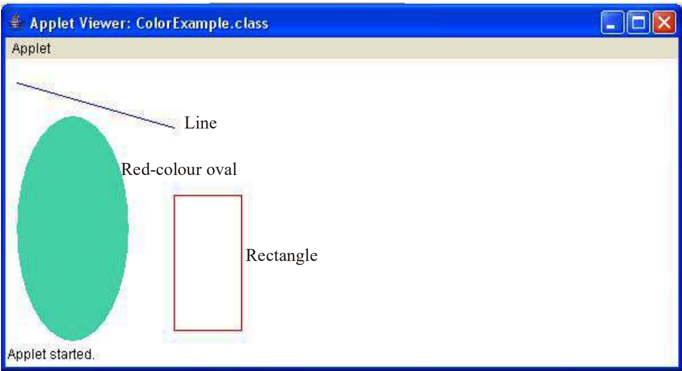
       //drawing a rectangle of red color
       gra.setColor(Color.red);
       gra.drawRect(150,120,60,120);
    }
}
```

The HTML code for `ColorExample` is as follows:

```
<HTML>
    <HEAD>
    </HEAD>
    <BODY>
       <CENTER>
          <APPLET
             CODE=”ColorExample.class”
             WIDTH    =600
             HEIGHT=250>
          </APPLET>
       </CENTER>
    </BODY>
</HTML>
```

NOTES

Output of the program:



Using `Font` Class

The `Font` class is used to apply different font styles to the text. To select or apply a new font, a `Font` object is required to be constructed.

The general form of the constructor of `Font` class is as follows:

```
Font(String font_name, int font_style, int font_size);
```

where, `font_name` is the name of the font, `font_style` is font style and `font_size` is the size of the font in points.

Some other methods of `Font` class are listed in Table 11.2.

Table 11.2 *Methods of `Font` Class*

Method	Description
`static Font getFont()`	Returns the currently selected font.
`int getSize()`	Returns the size of the font.
`String getName()`	Returns the name of the font.
`int getStyle()`	Returns the style of the font.
`String getFamily()`	Returns the name of the family of the font.

Program 8: An applet code to demonstrate the use of `Font` class.

```
import java.applet.*;
import java.awt.*;
public class FontExample extends Applet
{
    public void paint(Graphics gra)
    {
    Font MyFont=new Font("Courier New", Font.BOLD,16);
    gra.setFont(MyFont);
    gra.drawString("Hello Java",20,100);
```

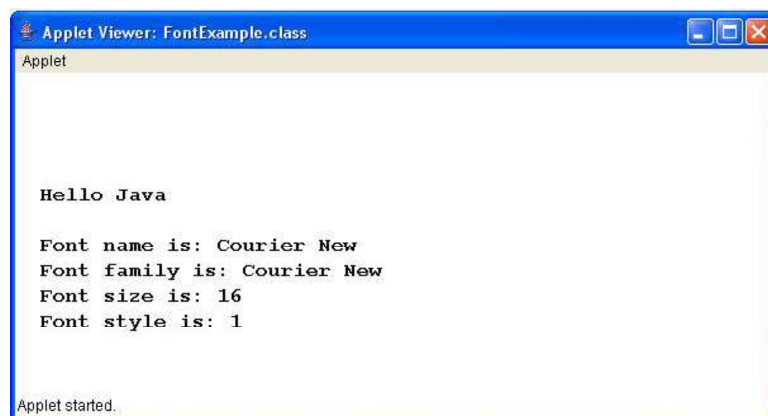
NOTES

```
    Font f=gra.getFont();
    String fontName=f.getName();
    gra.drawString("Font name is :"+fontName,20,140);
    String fontFamily=f.getFamily();
    gra.drawString("Font family is
:"+fontFamily,20,160);
    int fontSize=f.getSize();
    gra.drawString("Font size is :"+fontSize,20,180);
    int fontStyle=f.getStyle();
    gra.drawString("Font style is :"+fontStyle,20,200);
       }
}
```

The HTML code for `FontExample` is as follows:

```
<HTML>
    <HEAD>
    </HEAD>
    <BODY>
       <CENTER>
          <APPLET
             CODE="FontExample.class"
             WIDTH    =600
             HEIGHT=250>
          </APPLET>
       </CENTER>
    </BODY>
</HTML>
```

Output of the program:



NOTES

Using `Image` Class

The `Image` class is used to load and display images. To load an image, the `getImage()` method of the `Image` class is used and to display the image, the `drawImage()` method of the `Graphics` class is used.

The general form of the `getImage()` method is as follows:

```
Image getImage(URL pathname, String filename);
Image getImage(URL pathname);
```

where, `pathname` is the address of the image file on Web. When the image file and the source file are in the same directory, `getCodeBase()` method is used as first parameter to the method. `filename` is the name of the image file.

The general form of the `drawImage()` method is as follows:

```
boolean drawImage(Image image, int startx, int starty,
int width, int height, ImageObserver img_obj);
```

where, `image` is the image to be loaded in the applet, `startx` is the pixels space from the left corner of the screen, `starty` is the pixels space from the upper corner of the screen, `width` is the width of the image, `height` is the height of the image and `img_obj` is object of the class that implements `ImageObserver` interface.

Program 9: An applet code to demonstrate the use of `Image` class.

```
import java.applet.*;
import java.awt.*;
public class ImageExample extends Applet
{
private Image pic;
public void init()
{
pic=getImage(getCodeBase(), "java_pic.gif");
}
public void paint(Graphics gra)
{
gra.drawImage(pic,150,50,120,200,this);
    }
}
```

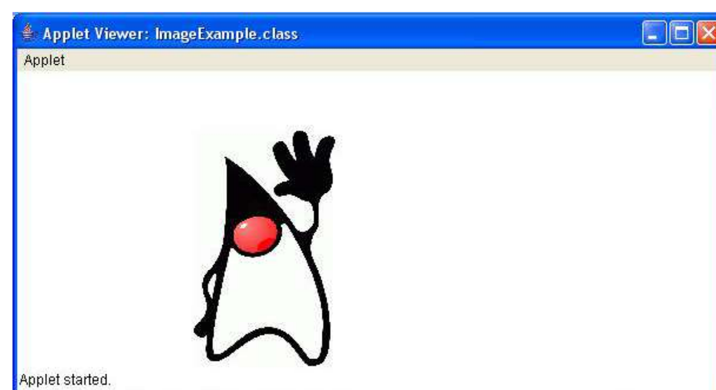
The HTML code for `ImageExample` is as follows:

```
<HTML>
    <HEAD>
```

```
    </HEAD>
    <BODY>
       <CENTER>
          <APPLET
             CODE="ImageExample.class"
             WIDTH    =600
             HEIGHT=250>
          </APPLET>
       </CENTER>
    </BODY>
</HTML>
```

NOTES

Output of the program:



Check Your Progress

9. What method is used to draw a line while working with graphics?
10. State the general form of `drawPolygon()` method?
11. What `Graphics` class provide to Java programmers?
12. What are the three primitive color provided by `Color` class?

11.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Error is an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing.

NOTES

2. **1. Run Time Error:** Run Time errors occur or we can say, are detected during the execution of the program. Sometimes these are discovered when the user enters an invalid data or data which is not relevant. Runtime errors occur when a program does not contain any syntax errors but asks the computer to do something that the computer is unable to reliably do. During compilation, the compiler has no technique to detect these kinds of errors. It is the JVM (Java Virtual Machine) which detects it while the program is running. To handle the error during the run time we can put our error code inside the try block and catch the error inside the catch block.

   2. Compile Time Error: Compile Time Errors are those errors which prevent the code from running because of an incorrect syntax such as a missing semicolon at the end of a statement or a missing bracket, class not found, etc. These errors are detected by the java compiler and an error message is displayed onto the screen while compiling. Compile Time Errors are sometimes also referred to as Syntax errors. These kind of errors are easy to spot and rectify because the java compiler finds them for you. The compiler will tell you which piece of code in the program got in trouble and its best guess as to what you did wrong. Usually, the compiler indicates the exact line where the error is, or sometimes the line just before it, however, if the problem is with incorrectly nested braces, the actual error may be at the beginning of the block. In effect, syntax errors represent grammatical errors in the use of the programming language.

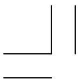
3. The term exception is an abbreviation for the phrase 'exceptional event'. It is an unpredicted event that occurs while the program is executing, and thus disrupts the normal flow of the program or terminates the program abnormally. Exception handling is used to provide solution to these exceptions, if a solution is not possible then to provide an error message alerting the user.

4. The code that must be executed whether an exception is thrown or not is kept within the `finally` block, which is optional.

5. `IOException` is thrown when an error occurs during input or output of data.

6. The syntax to define try-catch block is as follows:

```
try   //try block begins
{
   //code that may cause an exception
}  //try block ends
catch(exception_type ex)
{
   //code to handle the exception
}
```




NOTES

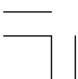
7. Sometimes, a method may generate an exception but does not have the appropriate exception handing mechanism for it. The methods that call such methods must be cautioned about this behavior so that calling methods can take appropriate measures to safeguard themselves against the exceptions. This is done by appending the `throws` keyword after the method name in the method declaration statement. The `throws` clause includes all types of exceptions excluding those belonging to `Error or Runtime` classes or their subclasses.
8 Java provides several built-in classes which define all types of exceptions. The `Throwable` class is the superclass and all the exception classes inherit methods defined by it. Two immediate subclasses of the `Throwable` class are the `Exception` class and `Error` class.
9. A line can be drawn using the `drawLine()` method of the `Graphics` class. This method takes four parameters, which represent the coordinates of the end points of the line.
10. The general form of the `drawPolygon()` method is

    ```
    void drawPolygon(int a[], int b[], int n);
    ```

    where, `a[]` is the array of integers having x-coordinates, `b[]` is the array of integers having y-coordinates and `n` is the total number of coordinate points required to draw a polygon.
11. The `Graphics` class provides different methods to draw and fill various shapes.
12. The `Color` class contains three primitive colors namely, red, blue and green, and all other colors are a combination of these three colors.

11.6 SUMMARY

- The term exception is an abbreviation for the phrase 'exceptional event'. It is an unpredicted event that occurs while the program is executing, and thus disrupts the normal flow of the program or terminates the program abnormally. Exception handling is used to provide solution to these exceptions, if a solution is not possible then to provide an error message alerting the user.
- The mechanism of exception handling in Java not only guarantees the detection and handling of run time errors but also provides a way to separate the error handling code from the rest of the program. Java exception handling is governed by the five keywords, namely, `try`, `catch`, `finally`, `throw` and `throws`.




NOTES

- The `Throwable` class is the superclass and all the exception classes inherit methods defined by it. Two immediate subclasses of the `Throwable` class are the `Exception` class and `Error` class.
- A set of statements that needs to be monitored for the exceptions is contained in the `try` block. Whenever an exception occurs within the `try` block, it is thrown. This passes the control to the `catch` block which handles the exception appropriately. The Java runtime system automatically throws system generated exceptions. If the user wants to throw the exception explicitly, the `throw` keyword is used. In case, multiple exceptions are thrown within a `try` block, Java allows the use of multiple `catch` blocks for handling all these exceptions.
- The `try` blocks can be nested, that is, one `try-catch` block can be placed inside another `try-catch` block. Sometimes, a method may throw exceptions which it cannot handle, i.e., it does not have the appropriate exception handing mechanism for the exception. This must be specified by using the `throws` keyword. The code that must be executed whether an exception is thrown or not is kept within the `finally` block, which is optional.
- User defined exceptions can be created by defining a subclass of `Exception` class and using the `throw` keyword.
- Java also allows the creation of an applet, which uses various classes to draw figures of different shapes, add images, display text in different fonts and styles, use and create various colors, etc. The `Graphics` class provides different methods to draw and fill various shapes.
- In `Graphics` class, `drawLine()` method is used to draw a line, `drawRect()` method is used to draw rectangle, `drawOval()` method is used to draw an ellipse, `drawArc()` method is used to draw an arc and `drawPolygon()` method is used to draw a polygon.
- The `Color` class provides various methods to use any color you want in the display. It defines the various color constants which can be directly used only by specifying the color of your choice. The `Color` class contains three primitive colors namely, `red`, `blue` and `green`, and all other colors are a combination of these three colors.
- To set the color of the background of an applet window, `setBackground()` method is used. Similarly, to set the foreground color to a specific color, that is, the color of text, `setForeground()` method is used.

- The `Font` class is used to apply different font styles to the text.
- The `Image` class is used to load and display images. To load an image, the `getImage()` method of the `Image` class is used and to display the image, the `drawImage()` method of the `Graphics` class is used.

NOTES

11.7 KEY WORDS

- **Exception handling:** Exception handling is used to provide solution to exceptions that occurs during runtime, if a solution is not possible then to provide an error message alerting the user
- **Applet:** Small program typically embedded within the Web page, which is used to create a dynamic and interactive application
- **`AppletViewer`:** A tool of the Java Development Kit, which is specifically designed for viewing applets

11.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What are the different types of errors in Java programming?
2. Define the use of `try` and `catch` blocks in exception handling.
3. How do you throw your own exception in Java?
4. Why is `finally` block used?
5. How do you apply different font styles in the `graphics` programming?
6. Elaborate the method of drawing and filling an ellipse and a circle.

Long-Answer Questions

1. Describe the various types of errors in Java.
2. Write a program illustrating the use of exception handling by using `try`, `catch`, `finally`, `throw` and `throws` blocks.
3. Differentiate between `throw` and `throws` keyword in exception handling.
4. List the various exception handling keywords.
5. Explain the process of drawing a rectangle in `graphics` programming.
6. Write a note on `graphics` class describing its various methods.

NOTES

11.9 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

UNIT 12 APPLET PROGRAMMING

NOTES

Structure

12.0 INTRODUCTION

Java applets were small applications written in the Java programming language, or another programming language that compiles to Java bytecode, and delivered to users in the form of Java bytecode. The user launched the Java applet from a web page, and the applet was then executed within a Java Virtual Machine (JVM) in a process separate from the web browser itself. A Java applet could appear in a frame of the web page, a new application window, Sun's AppletViewer, or a stand-alone tool for testing applets.

Java applets were introduced in the first version of the Java language, which was released in 1995. Java applets were usually written in Java, but other languages, such as Jython, JRuby, Pascal, Scala, or Eiffel (via SmartEiffel) can also be used. Since Java bytecode is cross-platform (or platform independent), Java applets can be executed by browsers (or other clients) for many platforms, including Microsoft Windows, FreeBSD, Unix, macOS and Linux. They cannot be run on modern mobile devices, which do not support Java.

The 'Java Applets' are used to provide interactive features to web applications that cannot be provided by HTML alone. They can capture mouse input and also have controls like buttons or check boxes. In response to user actions, an applet

can change the provided graphic content. An applet can also be a text area only; providing, for instance, a cross-platform command-line interface to some remote system. If needed, an applet can leave the dedicated area and run as a separate window.

NOTES

Fundamentally, an applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.

In this unit, you will study about the Java applet programming, how applets differ from applications, writing applets, building applet code, applet life cycle, creating an executable applet, designing a web page, applet tag, adding applet to HTML file, running the applet, passing parameters to applets, displaying numerical values and getting input from the user.

12.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the significance of applet programming in Java
- Understand how applets differ from applications
- Write applets, create an executable applet and build applet codes
- Elaborate on the applet life cycle
- Design a web page using applet tag
- Add applet to HTML file and run the applet
- Explain how parameters are passed to applets
- Display numerical values and get input from the user using Java applets

12.2 APPLETS

Applets are executed in the web browser and have made Java a web-enabled language. **Applet codes** also embed the HTML tags together. This is the package that has made Java a language for distributed application.

Generally, Java programs can be classified into two groups, namely applications and applets. Unlike applets, Java applications do not require a browser to run. They can be created like any normal programming language program. An applet is executed in the browser. An applet is a class file that displays graphics application in the web browser and one can embed applet codes in the web pages by HTML tags. Briefly, it can be said that an applet is a Java byte code embedded in an HTML page. The program structure of applets differs from the other Java applications.

NOTES

Definition

An **applet** is a dynamic and interactive program that can run inside a web page displayed by a Java-capable browser such as a HotJava Browser or an Internet Explorer browser, which are World Wide Web browsers used to view web pages. An applet is a class present in a java.applet package.

A special HTML tag is embedded in an applet to make it run on the web browser**.** The **appletviewer** application, present in the **jdk**, is used to run and check the applets. An applet has added advantages, such as frame, event-handing facility, graphics context and surrounding user interfaces.

Java applets have some restrictions to ensure full security and to make them virus free. Some of these are as follows:

(a) Applets have no permission to read or write the file system.

(b) Applets can communicate with the server in which they were stored originally, but not with the others.

(c) Applets cannot execute any programs on the system.

12.3 APPLET TAG

12.3.1 The Applet Tag

The body section of the HTML file contains a pair of `<APPLET...>` and `</APPLET>` tags, which allows one to provide the name of the applet and helps the browser recognize the space required for the applet. The following code shows the minimum code required to place the `FirstApplet` applet on the web page:

```
<APPLET
    CODE = FirstApplet.class
    WIDTH = 300
    HEIGHT = 150 >
</APPLET>
```

The above written HTML code helps a web browser load the compiled Java applet FirstApplet.class, which is present in the same directory as the HTML file. The `<APPLET>` tag discussed above includes the following:

- Name of the Applet
- Width of the Applet (in pixels)
- Height of the Applet (in pixels)

NOTES

12.3.2 The Applet Class

The applet class is a member of the Java Application Programming Interface (API) package, java.applet. The applet class is used for creating a Java program that displays an applet.

Figure 12.1 shows the hierarchical representation of the Java classes.

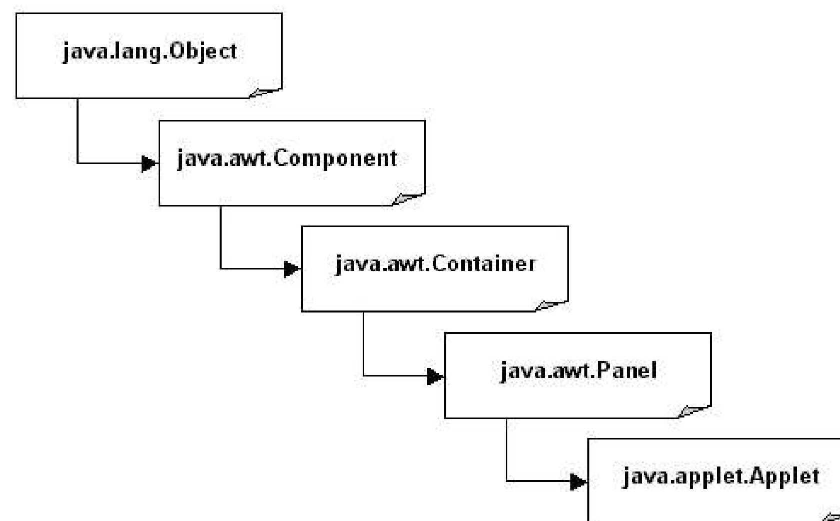


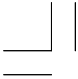
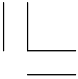
Fig. 12.1 *Java Class Hierarchies*

The applet class is extended from the class, Panel, which is further extended from the classes, Container, Component and Object. The object class is a member of the java.lang package and is called in the program automatically. The classes, Component, Container and Panel are members of the java.awt package and provide the visual components such as label, button or text fields. The applet is the only member of the java.applet package.

The applet class has several methods that are used to display the text and the image, play the audio file and respond when you interact with the applet. Table 12.1 lists the various methods of an applet class.

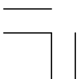
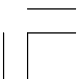
Table 12.1 *Applet Class Methods*

Methods	Description
`void destroy()`	Terminates an applet when the web browser or Java tool calls this method.
`getAccessibleContext()`	Returns the accessibility context of an object. Accessibility context represents the information about the accessible objects.
`getAppletContext()`	Returns the context that is associated with the applet.

NOTES

`GetAppletInfo()`	Returns a string type that describes the information about the applet.
`GetAudioClip(URL url, String clip-name)`	Returns an object of the audio clip that encapsulates the location and name of the audio clip.
`getCodeBase()`	Returns the URL that is associated with the invoking applet.
`getDocumentBase()`	Returns the URL of the HTML document that is invoking the applet.
`getImage(URL url, String image-name)`	Returns the image object that encapsulates the location and name of the image.
`getLocale()`	Returns the locale object that contains a set of end user preferences such as language, country, region or time.
`getParameter(String param-name)`	Returns a string that contains the parameter associated with the paramname.
`getParameterInfo()`	Returns a table that describes the information about the parameters that are recognized by the applet.
`void init()`	Begins the execution of the applet when it is called by the web browser or Java tool.
`IsActive()`	Returns the Boolean type true if the applet is started, otherwise returns false.
`void play(URL url, String clip-name)`	Plays the audio clip if it is found at the specified URL.
`void resize(int width, int height)`	Changes the size of an applet according to the specified height and width.
`void setStub(AppletStub stub-object)`	Returns the stub object of an applet. A stub is a piece of program that provides the linkage between the applet and the web browser.
`ShowStatus(String str)`	Displays a string in the status window of the appletviewer or web browser.

NOTES

void start()	Starts the execution of the applet when it is called by the web browser or Java tool.
void stop()	Suspends the execution of the applet when it is called by the web browser or Java tool. This suspended stage is resumed by the start() method.

Applets are programs based on the Internet that can be used to display text, image or animation. For example, an applet can be created that will display a text. To display text on the applet, the following four steps are to be followed:

1. Create a Java program for an applet
2. Compile the Java program
3. Create a web page that contains an applet
4. Run the applet

12.3.3 Adding an Applet to an HTML File

To run the applet, the HTML file that embeds an applet on the web page has to be created using the <APPLET> tag. This file has to be saved as FirstApplet.html. The following program code creates an HTML web page that embeds an applet:

```
<HTML>
   <HEAD>
      <TITLE>First Applet Program</TITLE>
   </HEAD>
   <BODY>
      <APPLET CODE = "FirstApplet.class" HEIGHT = 150
WIDTH = 300>
      </APPLET>
   </BODY>
</HTML>
```

The above code creates an HTML file in which the <APPLET> tag is used to embed the applet. The attribute of the <APPLET> tag, HEIGHT and WIDTH, sets the dimension of the applet window.

Note: You may specify the <APPLET> tag as a comment inside the Java applet source file. The code, which is specified inside the Java applet source file, is:

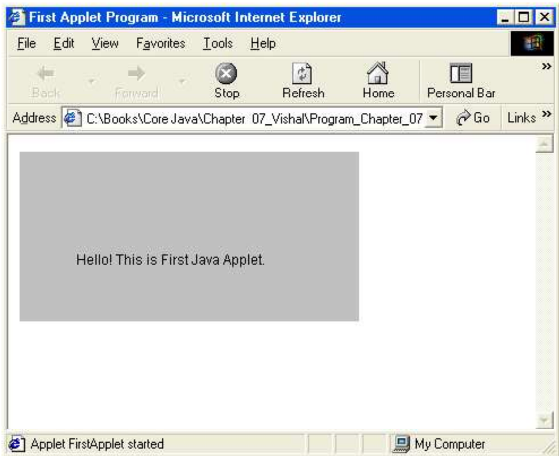
```
/*
<APPLET CODE = "FirstApplet" HEIGHT = 150 WIDTH = 300>
</APPLET>
*/
```

12.3.4 Running the Applet

You can run the applet either on the web browser or the appletviewer of Java. When you run the applet on a web browser, you invoke the HTML file from the web browser.

NOTES

The following screenshot displays the applet in Internet Explorer.



You can display the applet in appletviewer either by running the Java file, which consists of the <APPLET> tag as comment or by running the separate HTML file. The code to display the applet in the appletviewer of Java is:

```
C:\ >java>Unit 07>appletviewer FirstApplet.html
Or
C:\ >java>Unit 07>appletviewer FirstApplet.java
```

The above code shows an applet in the appletviewer of Java.

The following screenshot shows the output.



12.3.5 More About the Applet Tag

The <APPLET> tag is used to display an applet in the web browser or appletviewer tool of Java. You can also use the <OBJECT> tag of HTML to display the applet. The appletviewer executes the <APPLET> tag and displays an applet in an applet window. A web browser, such as Internet Explorer or Netscape Navigator, displays the applet on a web page interpreting the <APPLET> tag. The <APPLET> tag

NOTES

has various attributes that enable you to integrate your applet into the web page. The syntax of <APPLET> with all its attributes is:

```
<APPLET [CODEBASE] [CODE] [ALT] [NAME] [WIDTH] [HEIGHT]
[ALIGN] [VSPACE] [HSPACE]>
</APPLET>
```

The above syntax shows the <APPLET> tag and its attributes. The <APPLET> tag provides nine attributes, which are as follows:

- **CODEBASE:** Specifies the base URL of the applet class file. The base URL is the complete path of the applet where it is stored. This is an optional attribute that searches the executable file of an applet, class file, from the specified URL.
- **CODE:** Specifies the name of the file that contains a class file. This is a compulsory attribute of the <APPLET> tag that is relative to the URL of the HTML file.
- **ALT:** Specifies the tool tip text of an applet. This is also an optional attribute that is displayed to provide information about the applet whether the applet is running or not on the web browser.
- **NAME:** Specifies the name of an applet. This is an optional attribute of the <APPLET> tag. The name of an applet is obtained by the `getApplet()` method.
- **WIDTH:** Specifies the width of the applet display area in pixels.
- **HEIGHT:** Specifies the height of the applet display area in pixels.
- **ALIGN:** Specifies the alignment of the applet. The values of the ALIGN attribute are LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE and ABSBOTTOM.
- **VSPACE:** Sets the vertical space of the applet, in pixels, on each side of the applet.
- **HSPACE:** Sets the horizontal space of the applet, in pixels, on each side of the applet.

12.3.6 Passing Parameters to Applets

The <APPLET> tag enables you to pass the applet parameter in the applet using the <PARAM> tag. The <PARAM> tag has two attributes, NAME and VALUE, which are used to set the name and values of the parameter. The syntax of the <PARAM> tag that is enclosed within the <APPLET> tag is:

```
<APPLET>
<PARAM [NAME] [VALUE]>
</APPLET>
```

The above syntax shows the <APPLET> tag with parameter values. The NAME represents the name of the parameter and VALUE sets a value for that parameter. The <PARAM> tag is an empty tag. You can put several <PARAM>

tags inside a single <APPLET> tag. You can retrieve the value of a parameter using the `getParameter()` method that returns the specified parameter as a String object. The following program code shows an applet displaying the information of an end user that is passed as the parameter of the <PARAM> tag:

NOTES

Program 1: Displaying User Information by Passing Parameters

```
import java.awt.*;
import java.applet.Applet;
public class AppletHTML extends Applet
{
   String firstName, lastName, sex, add, city, phone,
email;
   Font font1, font2;
   public void init()
   {
      font1 = new Font("Arial", Font.PLAIN, 16);
      font2 = new Font("Arial", Font.BOLD, 20);
   }
   public void start()
   {
      //String param;
      firstName = getParameter("firstName");
      if(firstName == null)
      firstName = "Not Found";
      lastName = getParameter("lastName");
      if(lastName == null)
      lastName = "Not Found";
      sex = getParameter("sex");
      if(sex == null)
      sex = "Not Found";
      add = getParameter("add");
      if(add == null)
      add = "Not Found";
      city = getParameter("city");
      if(city == null)
      city = "Not Found";
      phone = getParameter("phone");
      if(phone == null)
      phone = "Not Found";
      email = getParameter("email");
      if(email == null)
      email = "Not Found";
   }
```

NOTES

```
    public void paint(Graphics g)
    {
       g.setFont(font2);
       g.drawString("-: User Information :- " , 75,
0);
       g.setFont(font1);
       g.drawString("First Name : " + firstName, 5,
0);
       g.drawString("Last Name : " + lastName, 5, 80);
       g.drawString("Sex : " + sex, 5, 100);
       g.drawString("Address : " + add, 5, 120);
       g.drawString("City : " + city, 5, 140);
       g.drawString("Phone No : " + phone, 5, 160);
       g.drawString("Email : " + email, 5, 180);
    }
/*
<APPLET CODE="AppletHTML" WIDTH=375 HEIGHT=200>
<PARAM NAME=firstName VALUE="Vishal">
<PARAM NAME=lastName VALUE="Jayaswal">
<PARAM NAME=sex VALUE="Male">
<PARAM NAME=add VALUE="779-A,Civil-Lines">
<PARAM NAME=city VALUE="Jhansi">
<PARAM NAME=phone VALUE="9891066098">
<PARAM NAME=email VALUE="vishal1431@rediffmail.com">
</APPLET>
*/
}
```

The above code shows the user information that is passed as a parameter of the <PARAM> tag. The `getParameter()` method retrieves the value of each parameter and displays this value on the applet.

Output of the program:



You can create an interactive or dynamic applet by passing the values of the parameter that are specified within the <PARAM> tag of the <HTML> document. When you are working with the <PARAM> tag, you can change the contents that are displayed on the applet by changing the parameter values in the HTML file.

12.4 APPLET LIFE CYCLE

Each applet class inherits the properties and methods of the class, Applet. An applet is loaded on the web browser or the appletviewer tool of Java. An Applet may change its current state when a specific method is called by AWT. Java provides four methods to change the state of an applet. These methods are as follows:

- `init()`
- `start()`
- `stop()`
- `destroy()`

The `init()` method is called when an applet loads the initialization process. The initialization process creates the objects that an applet needs for loading images, fonts and colours or for setting up the initial parameters such as variables and constants. This method is called only once when an applet is loaded the first time. The syntax to initialize the applet is:

```
public void init()
{
// init() method definition.
}
```

The `start()` method is called when an applet is initialized and starts the execution of the applet. The syntax to start the applet is:

```
public void start()
{
// start() method definition.
}
```

The `paint()` method is called when you want to display an applet. The syntax to display the applet is:

```
public void paint(Graphics g)
   {
      g.drawString(str, 10, 10);
   }
```

The `stop()` method is called when either the end user stops an applet or an applet loses the focus. The syntax to stop the applet is:

```
public void stop()
{
```

```
// stop() method definition.
}
```

NOTES

The `destroy()` method is called when an applet is destroyed. When you want to exit from the web browser or appletviewer tool of Java, an applet calls this method to free the resources. This method also occurs only once in the life cycle of an applet. The syntax to destroy the applet is:

```
public void destroy()
{
    // destroy() method definition.
}
```

Here, only the `paint()` method is a member of the graphics class, while the other methods are members of the applet class. Figure 12.2 shows the life cycle of an applet.

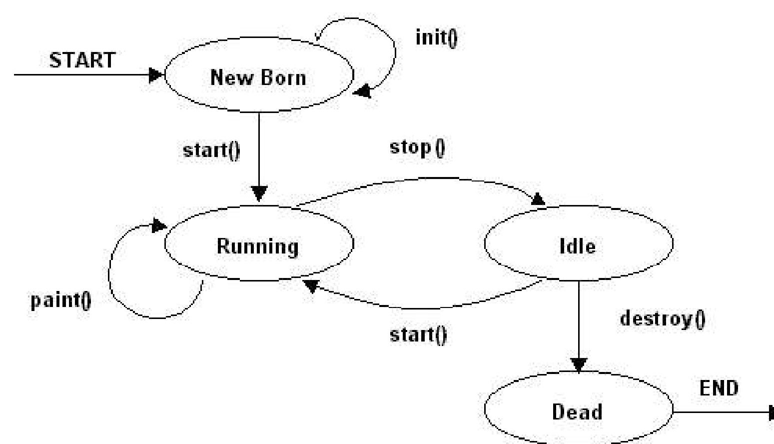


Fig. 12.2 *Life Cycle of an Applet*

There are four states in the life cycle of an applet: new born, running, idle and dead. The newborn state consists of a newly loaded applet that is initializing its resources. In the running state, an applet is executed and it displays the text or image that the applet contains. An applet that is initialized but is not currently in the running state is said to be in the idle state. When the `destroy()` method is invoked, the applet acquires a dead state and releases all the resources. The following code creates an applet that shows the various states of an applet life cycle:

Program 2: Creating an Applet showing the various States of an Applet Life cycle

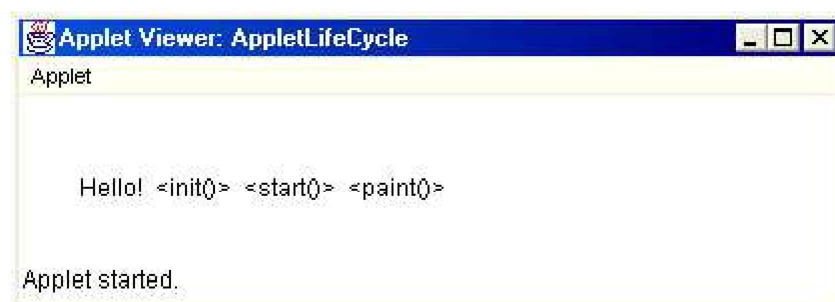
```
import java.applet.Applet;
import java.awt.Graphics;
public class AppletLifeCycle extends Applet
{
    String str = "Hello! ";
```

NOTES

```
    public void init()
    {
       str = str + " <init()> ";
    }
    public void start()
    {
       str = str + " <start()> ";
    }
    public void stop()
    {
       str = str + " <stop()> ";
    }
    public void destroy()
    {
       str = str + " <destroy()> ";
    }
    public void paint(Graphics g)
    {
       str = str + " <paint()> ";
       g.drawString(str, 30, 50);
    }
/*
<APPLET CODE="AppletLifeCycle" HEIGHT = 80 WIDTH = 410>
</APPLET>
*/
}
```

The above code shows the various states of the applet life cycle using applet methods. The `init()` method initiates all the variables that are used in the applet program. The `paint()` method displays text on the applet.

The following screenshot shows the output.



You can stop the execution of the applet by calling the `stop()` method.

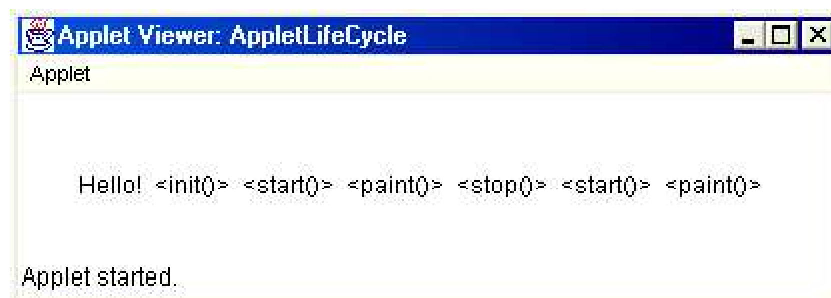
The following screenshot shows the suspended applet.

NOTES



AWT calls the `start()` and `paint()` methods when you re-start the applet.

The following screenshot shows re-starting the applet that was suspended by the `stop()` method.



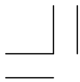
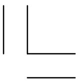
There is one additional method, `paint`, that is not a part of the applet life cycle but is used for displaying the contents of the applets on the appletviewer or web browser. The `paint()` method is called to display text or graphics on an applet. This method takes an argument, which is the instance of the Graphics class. The syntax to display the text on an applet is:

```
public void paint(Graphics g)
{
    // paint() method definitions.
}
```

The above syntax shows the `paint()` method of the graphics class of Java.

Check Your Progress

1. Define an applet.
2. What is the use of applet-viewer in applet programming?
3. What does <APPLET> tag refers to?
4. Write a code to add an applet to an HTML file.
5. How is CODEBASE used while programming applets?
6. What are the four methods use to change the state of an applet?

12.5 METHODS USING APPLETS

NOTES

An applet uses the classes and methods of AWT to perform its input and output operations. To display the output in the applet, one uses the `drawstring()` method present in the graphics class. Its general form is:

```
void drawString(String msg, int x_co, int y_co)
```

Here, msg holds the string to be written on the applet screen starting from the coordinates specified by int x_co, int y_co. In a Java window, the upper-left corner is location 0,0.

The `setBackground()` method is used to set the background colour of the applet.

The `setForeground()` method is used to set the foreground colour of the applet, i.e., the colour of the text to be written.

The above methods are defined in component class, and their general forms are:

(a) `void setBackground(Color Color)`

(b) `void setForeground(Color Color)`

Color specifies the new colour. The color class defines the following constants that can be used to specify colours:

```
Color.black Color.magenta
Color.blue Color.orange
Color.cyan Color.pink
Color.darkGray Color.red
Color.gray Color.white
Color.green Color.yellow
Color.lightGray
```

The following example sets the background colour to pink and the text colour to magenta.

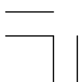
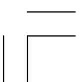
Program 3:

```
setBackground(Color.pink);
setForeground(Color.magenta);
```

An example for the creation of an applet is given below:

Program 4:

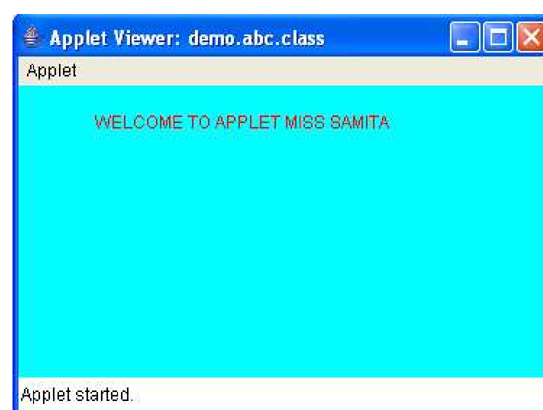
```
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleBanner" width=750 height=500>
</applet>
*/
```

NOTES

```
public class abc extends Applet {
String mesg = "WELCOME TO APPLET";
public void init() {
setBackground(Color.cyan);
setForeground(Color.red);
}
public void start() {
mesg = mesg+" MISS SAMITA";
}
public void stop() {
}
public void paint(Graphics g1) {
g1.drawString(mesg, 50, 30);
}
}
```

Output of the program:



From the above example, you can be seen that the steps to create an applet are:

First, one has to import both the packages, **java.applet** and **java.awt**. After that, the HTML tag has to be defined, which is required to make the applet run in the web browser. Then the `init()` method has to be defined. Then the `start()` method and the paint method have to be defined. Here, the `stop()` or `destroy()` methods are not required and therefore have not been defined.

Now from the above example, it can be seen that first the init() method is called and the background colour and foreground colour are set using the `setBackground(Color.cyan)` and `setForeground (Color.red)` methods. After that, the start method is called, in which the **mesg** variables content is modified. Then the paint method is called, in which the code `g.drawString(msg,50,30)` has been written to print the output on the applet.

NOTES

Another example would make the concept clearer and help in understanding the usage of passing parameters to the applet:



Program 5:

```
import java.applet.Applet;
import java .awt .*;
public class Second extends Applet
    {
    Font f = new Font (“TimesRoman”, Font.BOLD, 40);
    String name;
    public void init()
       {
       name = getParameter (“name”);
       if (name == null)
          {
          name = “Java”;
          name = “Have a nice day “ + name;
          }
       }
    public void paint(Graphics g1)
       {
       g1.setFont (f);
       g1.setColor (Color.blue);
       g1.drawString (name, 50, 50);
       }
    }
/*<applet code=”Second.class” width=200 height=200
align=TOP>
<param name=”name” value=”Sai”>
</applet>*/
```

An instance of the font class **f** is declared. This object has been initialized to contain TimesRoman as font name, font size as 40 and font style as BOLD. The `init()` method, declared in line 7, contains the `getParameter ()` method,

NOTES

which accepts the name (a string) as its parameter. The `paint ()` method of the component class is overridden to execute the paint () method in the class. This method contains the `drawString()` method, apart from the two methods, namely, `setFont()` and `setColor()`. These two methods are used to set the desired font and colour respectively.

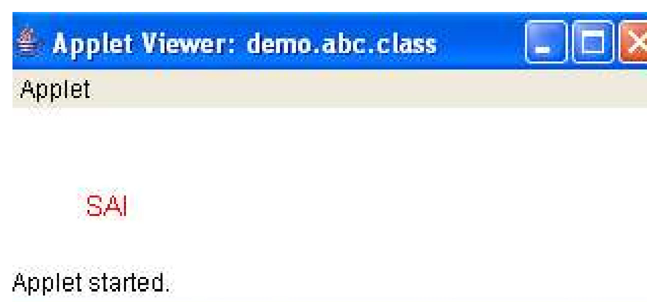
Passing Parameters to Applets

A programmer can set the parameter, as already explained. To retrieve the value, the `getParameter()` method has to be used, which takes the name of the parameter and returns the value stored in the parameter.

Program 6:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleBanner" width=750 height=500>
<param name="param1" value ="SAI">
</applet>
*/
public class abc extends Applet {
String mesg ;
public void init() {
setBackground(Color.white);
setForeground(Color.red);
}
public void start() {
mesg = getParameter("param1");
}
public void paint(Graphics g1) {
g1.drawString(mesg, 50, 30);
}
}
```

Output of the program:



Playing an Audio Clip

Consider the following example.

Program 7:

```
import java.applet.*;
import java.awt.*;
public class player extends Applet {
    AudioClip a1;
    public void init()
    {
       a1=getAudioClip(getCodeBase(),"sai.au");
    }
public void start()
{
a1.play();
}
public void paint(Graphics g1)
{
g1.drawString("playing music",50,30);
}
}
/*<applet code="Play" width=200 height=200>
</applet>*/
```

NOTES

Output of the program:



This program will open the applet and the music file named sai.au will be played.

NOTES

The `getAudioclip()` method returns the URL of the music file specified as its parameter value. Then the play method is called to play the music file.

One can only play the *.au format music files and the music file should reside in the same directory in which the Java file is present.

The following is a designer applet to play and stop music files:

Program 8:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class Play extends applet implements ActionListener
    {
    AudioClip a1;
    Button b1,b2;
       public void init()
       {
    b1=new Button("Start");
    b2=new Button("Stop");
       a1=getAudioClip(getCodeBase(),"sai.au");
       b1.addActionListener(this);
    b2.addActionListener(this);
    add(b1);
    add(b2);
       }
    public void actionPerformed(ActionEvent ae)
       {
       if(ae.getSource()==b1)
          {
          a1.play();
          }else{
          a1.stop();
          }
    }
    public void paint(Graphics g1)
       {
       g1.setColor(Color.red);
       g1.setFont(new Font("Arial",Font.BOLD,40));
       g1.drawString("This Is Music World",50,70);
       }
    }
```

```
/*<applet code="Play" width=200 height=200>
</applet>*/
```

Output of the program:



AppletContext

It is an interface that is implemented by an object that represents the environment of the applet. To use it, one has to first create an object of `AppletContext`, using the `getAppletContext()` method. This is shown below:

Program 9:

```
AppletContext context= getAppletContext( );
```

In AppletContext, some interesting methods are defined, such as the `getApplet()` method. It returns the name of the applet and the `getApplets( )` method, which returns all the applets in the document base.

```
getApplet( )
getApplets( )
```

In `AppletContext`, another method, `showDocument( )` is also defined, which either takes a URL or a URL and the file name in the form of a string. It creates an HTML page in the web browser, which shows the applet. It is useful with HTML frames where the applet should reside in a frame and the new HTML pages should be shown in another frame.

The `showStatus()` method shows the string passed to it as a parameter on the status bar at the bottom of the web browser. The following example shows the use of the `showDocument()` and `showStatus()` methods.

Program 10:

```
AppletContext Code Sample
import java.applet.*;
import java.awt.*;
import java.net.*;
   // <applet code="applet1.class" width=400 height=100></
applet>
   public class applet1 extends Applet{
   public void init(){
   try{
```

NOTES

```
    URL url1=new URL("http://wallpaper.net/pkomisar/
Flowere.html");
    getAppletContext( ).showDocument(url1);
    }catch(MalformedURLException me){}
    }
    public void paint( Graphics g1){
    g1.setFont(new Font("Monospaced", Font.BOLD, 22));
    g1.drawString("Hello World", 55, 65 );
    getAppletContext().showStatus("Puts the message in
status bar");
    }
    }
```

Now, in the above program, the user gets the current appletcontext using the `getAppletContext()` method. Then, using the `showDocument()` method, the user just transfers the control to another HTML file, which is passed to it as a parameter. The `showStatus()` method displays the string passed to it as a parameter at the bottom of the applet window.

Note: One can use console output in an applet, such as `System.out.println()`. The string passed to it will not get displayed on the screen; rather it will get displayed in the console. It is generally used for the purpose of debugging; otherwise, the use of these methods is discouraged.

Advantages of a Java Applet

A Java **applet** has the following advantages:

(a) The applet can work properly on all versions of installed Java, excluding the latest plug-in version.

(b) Almost all web browsers support the applet.

(c) A user can permit it to have full access to the machine on which it is running.

(d) It can improve with use, which means that after a first applet is run, the JVM is already running and starts quickly, benefitting the regular Java users. However, the JVM will need to restart each time the browser starts a fresh.

(e) In terms of its speed of execution, it is slower than C++ codes, but faster than JavaScript.

Disadvantages of a Java Applet

A Java applet has the following disadvantages:

(a) Sometimes, the Java plug-in is required, but it is not available on every web browser by default.

(b) The process of loading an applet is very slow.

It can now be concluded that without an applet, Java cannot get the honour of a web-enabled language. The AWT package present in Java is an important package. It is needed to develop a sophisticated applet.

12.6 SIMPLE AWT CONTROLS

NOTES

12.6.1 Introducing the Abstract Window Toolkit (AWT)

An end user interacts with applications via an interface. Java Development Kit (JDK) provides a package called the Abstract Window Toolkit (AWT), which enables you to design user-friendly interfaces. The AWT package is a collection of classes and methods that allow an end user to design and manage Graphical User Interface (GUI) based applications. AWT is used to support applet windows and it also helps in creating independent windows, such as frame windows or panel windows that run in a GUI environment.

Window Fundamentals

AWT is Java's largest package and can be included in a Java program by giving the java.awt.* statement. It is a part of Java Foundation Classes (JFC). AWT has 63 classes and 14 interfaces used for creating user interfaces. You can use AWT classes and interfaces to design and manage components such as lines, rectangles, ellipses, polygons, text boxes, command buttons, lists, menus and scroll bars.

The features of the AWT package are:

- It provides graphics and imaging tools that enable you to develop GUI.
- It provides layout managers that enable an end user to develop window layouts.
- It supports event handling.

The java.awt package contains all the classes used for developing graphical interfaces. These classes are organized in a hierarchical order. Figure 12.3 shows the hierarchy of classes in the java.awt package.

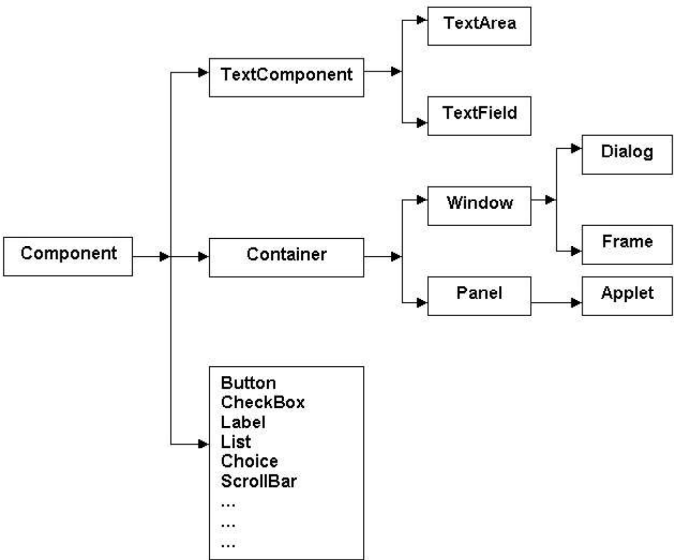


Fig. 12.3 *AWT Class Hierarchy*

NOTES

It shows the multi-level inheritance structure of AWT classes and subclasses. Multi-level inheritance means that a derived class has more than one base class, for example, the frame class is derived from the window class, which is derived from the container class. Component class and container class are the two important classes of AWT package. The component class consists of subclasses and methods for creating components such as command button, text boxes and labels. The container class enables you to create objects such as frames that contain components.

All other classes are subclasses of these two classes. Each subclass inherits the characteristics of its super class. Apart from inheriting the functionality of the upper class, it also has some additional functionality that is unique to its level.

Component Class

Component class is at the top of the AWT inheritance class hierarchy. It is the super class and all other classes in the hierarchy inherit its characteristics directly or indirectly. Component class is an abstract class that encapsulates all the properties of a visual component or object.

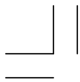
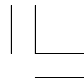
Component class defines the various methods that are used to handle events, define the layout of a component, provide the content of the component and repaint the window during the execution of a program. The methods that the objects of the components class use include:

- **void setSize (int width, int height):** Defines the size of a component in terms of width and size.
- **void setLocation (int x, int y):** Specifies the location of a component, in the window, at the position specified by the X and Y coordinates.
- **void setBounds (int x, int y, int width, int height):** Sets the bounds of a component in terms of position X & Y and size in terms of width and height.
- **void setForeground (Color B):** Sets the foreground colour of the component.
- **void setBackground (Color B):** Sets the background colour of the component.

Container Class

Container class is the subclass of the component class and its objects inherit all the characteristics of the component class. Therefore, container objects can also be used as component objects. The objects of container class are also known as containers. This class supports nested container objects. You can place one container object inside another container object, if required.

It is the function of the container object to determine the position of other component objects that it contains. Layout managers are responsible for performing this function.

NOTES

Containers use the `add()` function to nest other components inside them. The methods that are used to manipulate components on container objects are:

- **`void add (object o):`** Nests a component or an object inside a container.
- **`void remove (object o):`** Removes a component or an object from a container.
- **`void removeALL():`** Removes all the components or objects that are placed inside a container.

Panel Class

Panel is a subclass of the container class and is used for organizing components. The panel class inherits the methods from its super class rather than defining them. It uses the `add()` method to nest components inside it.

The panel class is a super class of the applet class. It does not contain a border, title bar or menu bar. It supports nested panel objects. A panel container object can have more than one panel and different panels can have different layouts.

In addition to the `add()` method, the methods that the panel class uses are:

- **`panel():`** Creates an object of the panel class.
- **`setLocation():`** Determines the position of a component inside the panel.
- **`setSize():`** Sets the size of a component in terms of width and height.
- **`setBounds():`** Sets the bounds of a component in terms of position and size.

Window Class

Window is another subclass of the container class. Its objects are called windows and they are top-level containers. Window objects are not contained in any other object.

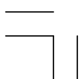
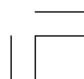
You can place window objects directly on the desktop. The window class has two subclasses, frames and dialog.

The methods used by the window class include:

- **`void pack():`** Adjusts the layout of a window according to the size and number of components it holds.
- **`void show():`** Displays a window on the screen and places the window on top of the other windows that may be open on the desktop.
- **`void dispose():`** Deallocates the memory allotted to a window.

Frame Class

Frame is a subclass of the window class. The frame class enables you to create pop-up windows. It is an important class of AWT that enables an end user to

create stand-alone applications. The frame window has a border, title bar and menu bar. The default layout for frame is border layout. Its objects respond to mouse, keyboard and focus events.

NOTES

Frame objects are created either through an applet program or through any other Java program using its constructor. If a frame object is created in an applet, it displays a message like Java Applet Window. Objects can be added to a frame using the `add ()` method.

Canvas Class

Canvas class is another type of window class and is used to create blank windows. Canvas objects are used only for drawing purposes, but they can enable an end user to create customized components. The canvas constructor takes no arguments and creates an empty drawing area. The canvas class does not define any method. It inherits the paint() method from the component class that needs to be overridden to provide the drawing functionality.

Working with Frames

Frame is the most often used window. End users can resize frames at runtime, and therefore, a frame object does not require X and Y dimensions to be declared at design time. Frames are resized after they have been created. Frames can be created using two types of constructors:

- **`Frame():`** Creates a standard window that does not contain any title bar. This constructor does not take parameters.
- **`Frame(String Title):`** Creates a window that has a title. This constructor accepts the title for a frame window as a string argument.

The title of the frame window can be changed using the `setTitle ()` method. The syntax to use the `setTitle ()` method is:

```
void setTitle (String newtitle)
```

In the above syntax, the `setTitle()` method changes the title of a frame by accepting the new title as an argument passed to it. The `Show()` method is used to make the frame window visible and active on the screen. The frame inherits the color class from its super classes. The color class sets the colour of the frame.

Creating a Frame

An instance of a frame can be created using any of the above two constructors.

You can use the frame objects unique methods as well as its inherited method by specifying the frame object name with the method name. The syntax to invoke a method is:

```
Frame1.show()
```

The following program code shows how to create a frame window:

Program 11:

How to Create a Frame

NOTES

```
/* Importing AWT package. */
import java.awt.*;
/* Creating a frame class. */
public class Framed1 extends Frame
{
  public static void main (String arg[])
   {
/* Creating a frame object. */
   Frame framed1 = new Frame("my");
   framed1.setSize(300,400);
   framed1.setVisible(true);
    }}
```

The above code shows how to create an empty frame. The `java.awt.*` package should be included in the Java program to implement the functions of the AWT classes in your program. The following screenshot shows the output.

Output of the program:



Setting Frame Properties

The frame class inherits all the methods of its super class. The inherited methods, such as `setSize()` and `setVisible()`, enable you to set the properties for a frame window. The default size of a frame is 0,0 in terms of width and height. You can set the size of a frame at any time in your program using the `setSize()` method. The syntax to implement the `setSize()` method is:

```
void setSize(int width, int height)
void setSize(Dimension size)
```

In the above syntax, you can specify the width and height of a frame window as arguments in the setSize() method. In addition, you can specify the dimensions of the frame window by passing an object of the dimension class. Size is the

object of the dimension class that has the width and height fields, which set the size of the frame in terms of pixels. These are the two ways of setting the frame window size property.

NOTES

You can use the `getSize()` method to obtain the current size of the frame. The syntax to obtain the current size of the frame window is:

```
Dimension getSize()
```

This method returns an object of dimension type. This object has the width and height fields that store the current size of the frame window, which is displayed on the screen. Other methods commonly used with the frame object are:

- **`setBackground():`** Sets the background colour of the frame.
- **`setEnabled():`** Enables or disables a frame window object.
- **`void setLocation(int x, int y):`** Sets the location of the frame to the left of the screen, by default, at X and Y positions.

Specifying the frame name followed by the method name sets the properties for a frame. The following program code shows how to set the properties of a frame window:

Program 12:

Setting the Properties of a Frame

```
import java.awt.*;
import java.awt.event.*;
class Framed extends Frame
{
  public static void main (String arg[])
   {
    Framed framd1 = new Framed();
    Framec fc = new Framec(framd1);
/* Setting frame properties. */
    framd1.setSize(250,250);
    framd1.setVisible(true);
    framd1.setTitle("Demo");
/* Setting the background colour for frame using color
object. */
    Color col = new Color(178,185,252);
    framd1.setBackground(Col);
/* Add windowlistener method to provide window
functionality. */
    framd1.addWindowListener(fc);
    }
}
```

```
/* Subclass of the windowadapter class is created to
provide window events such as closing a frame or minimizing
a frame. */
class Framec extends WindowAdapter
{
/* Object of the frame subclass is created. */
Framed fd;
/* Object of frame subclass is passed as arguments in the
windowadapter method. */
 Framec(Framed f)
 {  fd = f; }
/* Closing event is defined for frame. */
 public void windowClosing(WindowEvent we)
 {
  System.exit(0);
 }
}
```

NOTES

The above code shows how to create a frame window and set its properties using various methods such as `setSize()`, `setVisible()` and `setBackground()`. An object of the color class is created to set the background colour of the frame. An object of the windowadapter class is also created to provide all the functionalities of a window such as minimize, maximize and close.

Output of the program:



In this case, the frame size is set to 250,250, its visible property is set to true, its title is Demo and its colour is set to blue.

12.6.2 Creating AWT Controls

An AWT control is a component that allows the end user to interact with applications created in Java. All AWT controls in Java are subclasses of the component class. The component class provides the add() method to add AWT

components to containers such as applet windows. The various classes in Java for creating AWT controls are:

NOTES

- Label
- Button
- Check boxes
- Radio buttons
- Choice
- Text fields
- Text areas
- Scroll bars

All AWT components are capable of generating events other than labels, which are also known as passive AWT components. The AWT controls that can generate events are known as active AWT components.

Creating Labels

Labels are created in an applet to display text in a form. Labels can be used for naming other form controls such as text fields, checkboxes, radio buttons and text area boxes. The label class that is available in the java.awt package is used for creating labels. You use the label constructor to create an instance of the label class. The various constructors available in the label class are:

- `public Label( )`
- `public Label(String txt)`
- `public Label(String txt, int align)`

The `public Label()` constructor is an empty constructor that does not accept any parameter. It is used for creating empty labels that do not display any text.

The `public Label(String txt)` constructor accepts an argument of the string class. The string value passed to the constructor as an argument is displayed as the text message of the label class.

The `public Label(String txt, int align)` constructor accepts two arguments. The first argument is of the string class and represents the text message that the label displays. The second argument is an integer type value that describes the alignment of the label as left, right or centre.

Table 12.2 lists the parameter values for the integer argument in the constructor, public Label(String txt, int align).

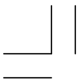



Table 12.2 *Alignment Values for a Label*

Values	Description
Label.RIGHT	Aligns label to the right of the applet window.
Label.CENTER	Aligns label to the centre of the applet window.
Label.LEFT	Aligns label to the left of the applet window.

NOTES

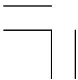
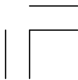
Note: Labels are aligned to the centre by default if the alignment is not specified.

You can use the methods available in the font and color classes to apply formatting and colour to the label objects. The following code shows how to create labels:

Program 13:

Creating Labels in an Applet Window

```
import java.awt.*;
import java.applet.*;
/*<applet code="LabelImpl.class" width="200"
height="300"></applet>*/
public class LabelImpl extends Applet
{
/*Creating Label references*/
Label l1,l2,l3;
public void init()
{
/*creating instances for labels*/
/*creating a right aligned Label*/
l1=new Label("Right Aligned Label",Label.RIGHT);
/*Creating a centre aligned Label*/
l2=new Label("Center Aligned Label",Label.CENTER);
/*Creating a left aligned Label*/
l3=new Label("Left Aligned Label",Label.LEFT);
/*Creating a Font object*/
Font f=new Font("Impact",Font.BOLD,25);
/*Creating objects of Color class*/
Color c1=new Color(255,200,210);
Color c2=new Color(200,200,250);
Color c3=new Color(100,100,250);
/*Applying Font Format to Label*/
l1.setFont(f);
l2.setFont(f);
l3.setFont(f);
/*Applying Colour Format to a Label*/
l1.setForeground(c1);
```

NOTES

```
l2.setForeground(c2);
l3.setForeground(c3);
/*Adding Labels to the Applet Window*/
add(l1);
add(l2);
add(l3);
}
}
```

The above code shows how to create labels that are right aligned, left aligned and centre aligned. Objects of the font and color classes are used in the code to apply font formats and colour to the labels. The `add()` method is used to add all the labels to the applet window.

Output of the program:



Java provides various methods with the label class that allow you to modify the existing labels in a program.

Table 12.3 describes the methods available in the Label class.

Table 12.3 *Methods of the Label Class*

Methods	**Description**
`void setText(String msg)`	Modifies the text of the label to the value passed as the argument String msg.
`String getText()`	Retrieves the string value of the label.
`void setAlignment(int align)`	Sets the alignment of the label to the value specified as argument.
`int getAlignment()`	Retrieves the current alignment of a label.

Creating Buttons

Buttons are the AWT controls that are used for handling events. Some of the actions associated with buttons are validating the form and submitting the information of the form to the database at the click of a button. Java provides the button class to create the AWT button components. The constructors of the button class for creating buttons are:

- `public Button()`
- `public Button(String txt)`

The `public Button()` constructor is the empty constructor for creating button controls. The button you create using this constructor does not contain a label. The `public Button(String txt)` constructor is the constructor for creating button controls with labels. The label of the button you create using this constructor is specified by the argument, String txt. You can enhance the appearance of the buttons you create using the font and color class objects. The following program code shows how to create buttons in an applet.

Program 14:

Creating Buttons

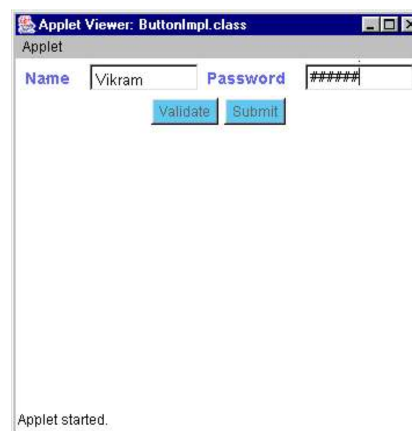
```
import java.awt.*;
import java.applet.*;
/*<applet code="ButtonImpl.class" width="200"
height="300"></applet>*/
public class ButtonImpl extends Applet
{
Label l1,l2;
TextField t1,t2,t3;
/*Creating references of the Button class*/
Button b1,b2;
public void init()
{
l1=new Label("Name");
l2=new Label("Password");
/*Creating Password and Text Fields*/
t1=new TextField(10);
t2=new TextField(10);
t2.setEchoChar('#');
/*Creating instances of Button class*/
b1=new Button("Validate");
b2=new Button("Submit");
/*Creating an instance of the Font class*/
Font f=new Font("Arial",Font.BOLD,14);
```

NOTES

```
Font f1=new Font("Arial",Font.BOLD | Font.ITALIC,12);
Color c=new Color(100,100,250);
Color bc=new Color(100,200,250);
Color fc=new Color(100,100,90);
l1.setFont(f);
l2.setFont(f);
/*Applying colour to AWT components*/
l1.setForeground(c);
l2.setForeground(c);
b1.setForeground(fc);
b2.setForeground(fc);
b1.setBackground(bc);
b2.setBackground(bc);
/*Adding the AWT Controls*/
add(l1);
add(t1);
add(l2);
add(t2);
add(b1);
add(b2);
}
}
```

The above code shows how to create buttons using constructors of the button class. You can add colour to the background and foreground of the button objects using instances of the color class. You can apply formatting to the label font of the label objects using an instance of the color class.

Output of the program:



NOTES

The button class provides methods that allow you to modify the properties of the existing button objects. Table 12.4 describes the two methods of the button class.

Table 12.4 *Methods of the Button Class*

Methods	Description
`void setLabel(String txt)`	Modifies the existing label of the button to the value specified by the argument, String txt.
`String getLabel()`	Retrieves the value of the label of the button.

Creating Checkboxes and Radio Buttons

The checkbox control is used for making multiple selections. Java provides the checkbox class for creating check boxes. The checkbox class is also used for creating radio buttons using the class, CheckboxGroup. The constructors of the checkbox class used for creating checkbox and radio button objects are:

- `public Checkbox()`
- `public Checkbox(String txt)`
- `public Checkbox(String txt, boolean true_false)`
- `public Checkbox(String txt, boolean tf, CheckboxGroup cbg)`
- `public Checkbox(String txt, CheckboxGroup cbg, boolean tf)`

The constructor, `public Checkbox()` is an empty constructor. This constructor creates a checkbox without a label. The constructor, `public Checkbox(String txt)` creates a checkbox with a label. The value of the label is specified by the argument String txt.

The constructor, `public Checkbox(String txt, boolean true_false)` creates a checkbox with a label. You can also specify whether the checkbox initially appears selected or deselected. The value of the label is specified by the argument, String txt. If the value of the argument, boolean true_false is true then the checkbox appears selected. If the value of the argument, boolean true_false is false then the checkbox appears deselected.

The `public Checkbox(String txt, boolean tf, CheckboxGroup cbg)` is the constructor for creating radio buttons. The argument, String txt specifies the label of the radio button. The argument, boolean tf specifies whether the radio button appears selected by default or not. The argument, `CheckboxGroup cbg`, is an instance of the `CheckboxGroup` class that specifies several radio buttons, which belong to the same group. You can select only one radio button in each group.

NOTES

The constructor, `public Checkbox(String txt, CheckboxGroup cbg, boolean tf)` creates radio buttons. This constructor is the same as public Checkbox(String txt, boolean tf, CheckboxGroup cbg). Only the order of the arguments, CheckboxGroup cbg and boolean tf, is interchanged.

The following program code shows how to create checkboxes and radio buttons in an applet:

Program 15:

Creating Checkboxes and Radio Buttons

```
import java.awt.*;
import java.applet.*;
/*<applet code="CheckboxImpl.class" width="200"
height="300"></applet>*/
public class CheckboxImpl extends Applet
{
Label l1,l2,l3,l4;
TextField t1,t2;
Button b1;
/*Creating references for Checkbox class*/
Checkbox c1,c2,c3,c4,c5;
Checkbox r1,r2;
/*Creating references for Checkbox class*/
CheckboxGroup cbg;
public void init()
{
l1=new Label("Name");
l2=new Label("Password");
l3=new Label("Hobby");
l4=new Label("Gender");
t1=new TextField(10);
t2=new TextField(10);
t2.setEchoChar('*');
/*Creating instances of the Checkbox class*/
c1=new Checkbox("Sports");
c2=new Checkbox("Music");
c3=new Checkbox("Reading", true);
c4=new Checkbox("Dancing");
c5=new Checkbox("Gardening");
/*Creating instance of the CheckboxGroup class*/
cbg=new CheckboxGroup();
r1=new Checkbox("Female",true,cbg);
```

```
r2=new Checkbox("Male",false,cbg);
/* Creating instance of Button class*/
b1=new Button("Submit");
/* Creating instance of Font class*/
Font f=new Font("Arial",Font.BOLD,14);
/* Creating instance of the Color class*/
Color c=new Color(100,100,250);
/*Applying font and colour formats to the Labels*/
l1.setFont(f);
l2.setFont(f);
l3.setFont(f);
l4.setFont(f);
l1.setForeground(c);
l2.setForeground(c);
l3.setForeground(c);
l4.setForeground(c);
/*Adding AWT components*/
add(l1);
add(t1);
add(l2);
add(t2);
add(l3);
add(c1);
add(c2);
add(c3);
add(c4);
add(c5);
add(l4);
add(r1);
add(r2);
add(b1);
}
}
```

The above code shows how to create checkboxes and radio buttons using the constructors of the checkbox class. An instance of the `CheckboxGroup class` is created to group the radio buttons. The `add()` method is used for adding checkboxes and radio buttons to the applet window.

Output of the program:

The checkbox class provides methods that allow you to modify the properties of the existing checkboxes and radio buttons.

NOTES

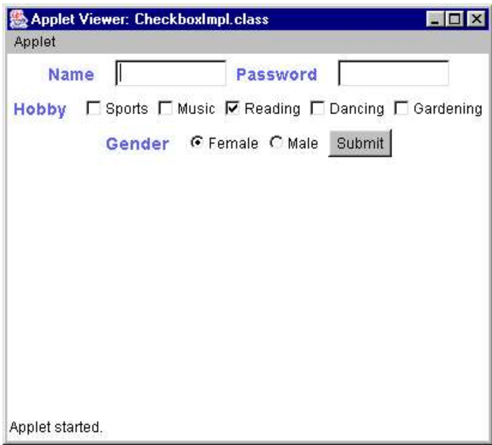


Table 12.5 describes the methods of the checkbox class.

Table 12.5 *Methods of the Checkbox Class*

Methods	Description
`boolean getState()`	Retrieves a boolean value that represents whether the radio button or checkbox is selected or not. A true value represents that the checkbox or radio button control is selected. A false value represents that the control is not selected.
`void setState(boolean tf)`	Modifies the state of the radio button or checkbox to either selected or deselected. If the value of the argument, boolean tf, is true then the control appears selected. If the value of the argument, boolean tf, is false then the control appears deselected.
`String getLabel()`	Retrieves the label of the radio button or checkbox.
`void setLabel(String str)`	Modifies the label of the radio button or checkbox to the value specified by the argument, String str.
`Checkbox getSelectedCheckbox()`	Retrieves the selected radio button object from a group.
`void setSelectedCheckbox(Checkbox rd)`	Selects a radio button specified by the argument, Checkbox rd.

Creating Choice Controls

You can create combination boxes that are also known as drop-down menus using the class, Choice, in Java. The choice class contains a single constructor,

public `Choice()`, which creates an instance of the choice class. Then, the `add()` method of the choice class is used for adding members to the combination box. The syntax for creating a combination box using the choice class is:

```
Choice ch=new Choice();
ch.add("Item 1");
ch.add("Item 2");
ch.add("Item 3");
add(ch);
```

The above syntax shows that the `add()` method of the choice class accepts a string type argument, which is added to the choice object. Another `add()` method is also used for adding the choice object to the applet window. The following program code shows how to create a combination box using the choice class:

Program 16:

Creating Combination Boxes

```
import java.awt.*;
import java.applet.*;
/*<applet code="ChoiceImpl.class" width="200"
height="300"></applet>*/
/*Creating class to display Choice options*/
public class ChoiceImpl extends Applet
{
Label l1,l2;
Choice ch1,ch2;
public void init()
{
/* Creating instances of the Label class*/
l1=new Label("Select a SAARC Nation");
l2=new Label("Select your country");
/* Creating instances of the Choice class*/
ch1=new Choice();
ch2=new Choice();
/* Creating instances of the Font class*/
Font f1=new Font("Arial",Font.BOLD | Font.ITALIC,14);
/* Creating instances of the Color class*/
Color c=new Color(100,100,250);
/* Applying font and colour formats to the components*/
l1.setFont(f1);
l2.setFont(f1);
l1.setForeground(c);
l2.setForeground(c);
/*Adding Items to the first Choice instance*/
```

NOTES

```
ch1.add("Bangladesh");
ch1.add("Bhutan");
ch1.add("India");
ch1.add("Maldives");
ch1.add("Nepal");
ch1.add("Pakistan");
ch1.add("Sri Lanka");
/*Adding Items to the second Choice instance*/
ch2.add("India");
ch2.add("U.S.A");
ch2.add("U.K");
ch2.add("Japan");
ch2.add("China");
ch2.add("Pakistan");
ch2.add("Korea");
/*Adding AWT components*/
add(l2);
add(ch2);
add(l1);
add(ch1);
}
}
```

The above code shows how to create two combination boxes. One combination box contains the name of the seven SAARC nations and the other combination box contains the names of some countries. You create two instances of the choice class that contain this information.

The choice class provides methods that allow you to modify the properties of the combination boxes.

Output of the program:



Table 12.6 describes the methods of the Choice class.

Table 12.6 Methods of the Choice Class

Methods	Description
`String getSelectedItem()`	Retrieves the text option from the combination box that is selected by the end user.
`int getSelectedIndex()`	Retrieves the position of the text member in the combination box that is selected by the end user.
`int getItemCount()`	Retrieves the total number of members in the combination box.
`void select(int indx)`	Selects the text option specified at the location by the argument, int indx.
`String getItem(int indx)`	Retrieves the member that exists at the position specified by the argument, int indx.

NOTES

Creating Text Fields

A text field is an AWT component that allows you to accept input from the end user. Java provides the textField class to create a text field control. The textField class is available in the java.awt package. The various constructors of the textField class are:

- `public TextField()`
- `public TextField(int num)`
- `public TextField(String txt)`
- `public TextField(String txt, int num)`

The `public TextField()` constructor is an empty constructor. It is used for creating text fields with default settings. You make use of empty text fields to create form controls.

The `public TextField(int num)` constructor creates a text field of the size specified by the argument, int num. The argument, num, is an integer type value that specifies the number of characters as the length of the text field. For example, the value of int num is set to 10 for creating a text field, which can hold 10 characters.

The `public TextField(String txt)` constructor creates a text field where a string message is displayed within the text field. The value of the argument String txt is displayed in the text field as the default text.

The `public TextField(String txt, int num)` constructor creates a text field, which is of the length specified by the argument, int num. The value specified as the argument, String txt, appears as the default text within the text field.

The following program code shows how to create text fields in an applet using the `TextField` class:

Program 17:

NOTES

Creating Text Fields

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="TextArea" width=50 height=800>
  </applet>
*/
public class TextArea extends Applet
  implements ActionListener {
    TextField tfname, tlname, tstreet, tstate, tpin, templovee;
  public void init() {
    Label fname = new Label("First Name: ", Label.RIGHT);
    Label lname = new Label("Last Name: ", Label.RIGHT);
    Label street = new Label("Street: ", Label.RIGHT);
    Label state = new Label("State: ", Label.RIGHT);
    Label pin = new Label("Pin: ", Label.RIGHT);
      Label employee = new Label("Employee Code: ", Label.RIGHT);
    tfname = new TextField(12);
    tlname = new TextField(12);
    tstreet = new TextField(12);
    tstate = new TextField(12);
    tpin = new TextField(12);
    templovee = new TextField(12);
    add(fname);
    add(tfname);
    add(lname);
    add(tlname);
    add(street);
    add(tstreet);
    add(state);
    add(tstate);
    add(pin);
    add(tpin);
    add(employee);
    add(templovee);
```

NOTES

```
   // register to receive action events
   tfname.addActionListener(this);
   tlname.addActionListener(this);
   tstreet.addActionListener(this);
   tstate.addActionListener(this);
   tpin.addActionListener(this);
   temployee.addActionListener(this);  }
  // User pressed Enter.
  public void actionPerformed(ActionEvent ae) {
    repaint();
  }
  public void paint(Graphics g) {
     g.drawString("First Name: " + tfname.getText(), 5,
260);
     g.drawString("Last Name: " + tlname.getText(), 5,
280);
       g.drawString("Street: " + tstreet.getText(), 5,
300);
    g.drawString("State: " + tstate.getText(), 5, 340);
    g.drawString("Pin: " + tpin.getText(), 5, 360);
    g.drawString("Employee Code: " + temployee.getText(),
5, 380);
}
}
```

The above code shows how to create text fields using the textField class. The add() method is used for adding the text field to the applet window.

Output of the program:



The textField class also contains methods that are used for modifying the properties of the existing text fields.

Table 12.7 describes the methods of the textField class.

NOTES

Table 12.7 *Methods of the Text Field Class*

Methods	Description
`String getText()`	Retrieves the text input in the text field.
`void setText(String txt)`	Sets the value in the text field to the value assigned to the argument, String txt.
`String getSelectedText()`	Retrieves the selected text from the text field. The end user selects the text either by using the mouse or the keyboard.
`void select(int start, int end)`	Selects the text within the text field starting from the position specified by the argument, int start to the position specified by the argument, int end.
`boolean isEditable()`	Retrieves a value, true or false that tells whether the end user can modify the text within the text field or not. A true value means that the text within the text field can be modified. A false value means that the value within the text field cannot be modified.
`void setEditable(boolean true_false)`	Assigns the value true or false that specifies the editable property of the text field.
`void setEchoChar(char encchar)`	Converts a text field to a password field. The characters entered by the end user appear encrypted by the character specified by the argument, char encchar.
`Boolean echoCharIsSet()`	Retrieves a value, true or false that determines whether the text field is converted to a password field or not.
`char getEchoChar()`	Retrieves the character that encrypts the text in a password field.

You can create a password field using the methods of the textField class. The following program code shows how to create password fields:

Program 18:

Creating Password Fields

NOTES

```
import java.awt.*;
import java.applet.*;
import java.util.*;
/*<applet code="PasswordImpl.class" width="200"
height="300"></applet>*/
public class PasswordImpl extends Applet
{
Label l1,l2,l3;
TextField t1,t2,t3;
public void init()
{
/*Creating instances of Label*/
l1=new Label("Name",Label.LEFT);
l2=new Label("Password",Label.LEFT);
l3=new Label("Date",Label.LEFT);
/*Creating instances of TextField*/
t1=new TextField(25);
t2=new TextField(25);
t3=new TextField(25);
/*Creating an instance of the Date class */
Date d=new Date();
/*Creating an instance of the Font class */
Font f=new Font("Arial",Font.BOLD,12);
/*Creating an instance of the Color class */
Color c=new Color(100,200,250);
/*Applying fonts*/
l1.setFont(f);
l2.setFont(f);
l3.setFont(f);
/*Applying colour*/
l1.setForeground(c);
l2.setForeground(c);
l3.setForeground(c);
/*Adding AWT controls to the applet window*/
add(l3);
add(t3);
add(l1);
add(t1);
add(l2);
```

NOTES

```
add(t2);
/*Converting text field to the password field*/
t2.setEchoChar('*');
/*Displaying current date in a text field*/
t3.setText(d.toString());
/*Setting the editable property of a text field to false*/
t3.setEditable(false);
}
}
```

The above code shows how to create a password field. An instance of the date class is created to display the current date in a text field. The text field that displays the current date is disabled using the `setEditable()` method.

Output of the Program:



Creating Text Areas

A multi-line text field is called a text area box. A text field accepts a single line of input whereas a text area box can accept several lines of input. Java provides the AWT class, TextArea to create a text area box. The constructors for creating a text area box are:

- `public TextArea()`
- `public TextArea(int height_in_chars, int width_in_chars)`
- `public TextArea(String txt)`

- `public TextArea(String txt, int height_in_chars, int width_in_chars)`
- `public TextArea(String txt, height_in_chars, int width_in_chars ,int scroll_value)`

The constructor, `public TextArea()` is the empty constructor. It creates a text area box with the default settings for width and height.

The constructor, `public TextArea(int height_in_chars, int width_in_chars)` allows you to specify the height and width of the text area box. The argument int `height_in_chars` specifies the height of the text area box in characters. The argument int width_in_chars specifies the width of the text area box in characters.

The constructor, `public TextArea(String txt)` allows you to specify the default text within the text area box. The height and width of the text area box are set to default size.

The constructor, `public TextArea(String txt, int height_in_chars, int width_in_chars)` allows you to specify the default text, height and width of the text area box. The argument String txt specifies the default text within the text area box. The argument, `int height_in_chars`, specifies the height of the text area box in characters. The argument, `int width_in_chars`, specifies the width of the text area box in characters.

The constructor, `public TextArea(String txt, int height_in_chars, int width_in_chars, int scroll_value)` allows you to specify the default text, height, width and scroll bars of the text area box. The argument String txt specifies the default text within the text area box. The argument `int height_in_chars` specifies the height of the text area box in characters. The argument `int width_in_chars` specifies the width of the text area box in characters. The argument `int scroll_value` specifies the type of scrollbar in the text area box. Table 12.8 lists the scrollbar values for the text area box that can be specified with the `TextArea()` constructor.

Table 12.8 *Scrollbar Values*

Values	Description
`SCROLLBARS_BOTH`	Displays both the horizontal and the vertical scrollbars with the text area box.
`SCROLLBARS_ HORIZONTAL_ONLY`	Displays the horizontal scrollbar with the text area box.
`SCROLLBARS_NONE`	Displays no scrollbars.
`SCROLLBARS_VERTICAL_ONLY`	Displays the vertical scrollbar with the text area box.

You may use the textArea constructors to create text area boxes. The following program code shows how to create text area boxes using the various constructors available with the textArea class:

NOTES

Program 19:

Creating Text Area Boxes

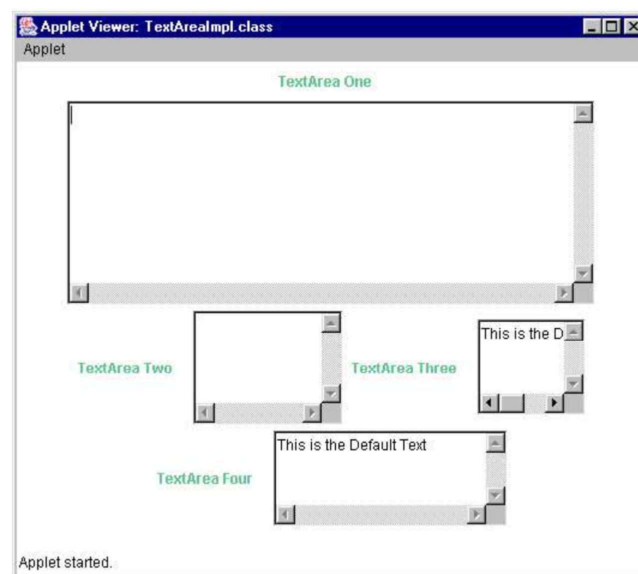
```
import java.awt.*;
import java.applet.*;
/*<applet code="TextAreaImpl.class" width="200"
height="300"></applet>*/
public class TextAreaImpl extends Applet
{
Label l1,l2,l3,l4;
/*Creating references of the TextArea class*/
TextArea ta1,ta2,ta3,ta4;
public void init()
{
/*creating instances for Label class*/
l1=new Label("TextArea One");
l2=new Label("TextArea Two");
l3=new Label("TextArea Three");
l4=new Label("TextArea Four");
/*Creating instances for the TextArea class*/
ta1=new TextArea();
ta2=new TextArea(5,15);
ta3=new TextArea("This is the Default Text",4,10);
ta4=new TextArea("This is the Default
Text",4,25,TextArea.SCROLLBARS_BOTH);
/*Creating instance of the Font class*/
Font f=new Font("Arial",Font.BOLD,12);
Color c=new Color(100,200,150);
l1.setFont(f);
l2.setFont(f);
l3.setFont(f);
l4.setFont(f);
/*Applying colour to the labels*/
l1.setForeground(c);
l2.setForeground(c);
```

NOTES

```
l3.setForeground(c);
l4.setForeground(c);
/*Adding AWT components */
add(l1);
add(ta1);
add(l2);
add(ta2);
add(l3);
add(ta3);
add(l4);
add(ta4);
}
}
```

The above code shows how to create several text area boxes using the constructors of the `TextArea class`. The `add()` method is used for adding the text area boxes to the applet window.

Output of the program:



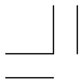
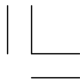
The `TextArea` class provides methods that allow you to modify the properties of an existing text area box. Both the `TextField` and the `TextArea` classes extend the `textComponent class` and therefore, all the methods available in the textComponent class are common to both the `TextArea` and `TextField` classes.

NOTES

Table 12.9 lists the methods that you can use with the `TextArea` class.

Table 12.9 *Methods for the TextArea Class*

Methods	Description
`String getText()`	Retrieves the text input in the text area box.
`void setText(String txt)`	Sets the string value in the text area box to the value assigned to the argument, String txt.
`String getSelectedText()`	Retrieves the selected text from the text area box. The end user selects the text either by using the mouse or the keyboard.
`void select(int start, int end)`	Selects the text within the text area box starting from the position specified by the argument, int start, to the position specified by the argument, int end.
`boolean isEditable()`	Retrieves a value, true or false that tells whether the end user can modify the text within the text area box or not. A true value means that the text within the text area box can be modified. A false value means that the text within the text area box cannot be modified.
`void setEditable(boolean true_false)`	Assigns the value, true or false that specifies the editable property of the text area box.
`void append(String txt)`	Appends the text specified in the argument, `String txt`, to the end of the existing text in the text area box.
`void insert(String txt, int index)`	Inserts a string specified by the argument, String txt, at the position specified by the argument, `int index`.
`void replaceRange (String txt, int start, int end)`	Replaces the text starting from the position specified by the argument, int start, to the position specified by int end with the text specified by String txt.

Creating Scrollbars

NOTES

Scrollbars allow end users to select multiple options from the list continuously. Scrollbars are used when all the options in the list box cannot be displayed on the screen at the same time. The scrollbars allow the end user to select the options that are hidden. To select the hidden options, you first need to move the scrollbars to display the options on the screen and then select the options. Scrollbars can be horizontal or vertical.

Horizontal scrollbars are generally used to display the continuous information in a text area, where the text is present beyond the width of the text area. Vertical scrollbars are used to present multiple options in a text box, where multiple options are present beyond the total length of the list box in which multiple options are present. You must have noticed vertical scrollbars on web-based forms, where the end user needs to select an option regarding the country out of multiple options of the country present in the list box.

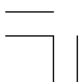
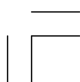
A scroll bar consists of small arrows at the end that are used to move the scroll bar in the corresponding direction that can be horizontal or vertical. To move the scroll bar you need to click the arrow. Each time an arrow is clicked the scroll bar moves one point up or down in the case of a vertical scrollbar and sideways in the case of a horizontal scroll bar.

The scroll bar also consists of a slider that assists in moving the scroll bar. To move the scroll bar using the slider, you need to press the slider and drag the mouse to move the slider in the desired direction without leaving the mouse. The slider bar also helps in determining the current value of the scroll bars in the list box. Whenever a user drags the slider or clicks the arrows present at the corner, the change in the value of the scrollbars is reflected by the slider. You can use list boxes to visualize the effects of scrollbars. The following program code shows how to create scrollbars in Java:

Program 20:

Creating Lists

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="Scroll" width=300 height=200>
  </applet>
*/
```

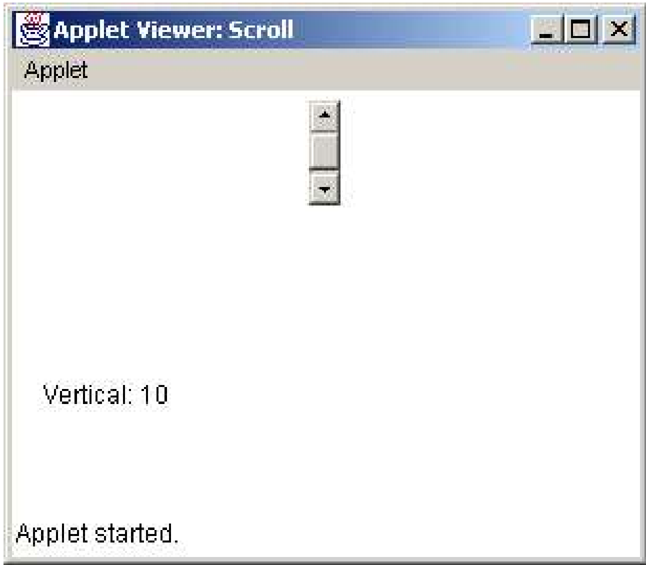
NOTES

```
public class Scroll extends Applet
  implements AdjustmentListener, MouseMotionListener {
  String message = "";
  Scrollbar vertical;
  public void init() {
    int h = Integer.parseInt(getParameter("height"));
    vertical = new Scrollbar(Scrollbar.VERTICAL,
                             10, 150, 10, h);
    add(vertical);
    // register to receive adjustment events
    vertical.addAdjustmentListener(this);
    addMouseMotionListener(this);
  }
  public void adjustmentValueChanged(AdjustmentEvent ae)
{
    repaint();
  }
  // Update scroll bars to reflect mouse dragging.
  public void mouseDragged(MouseEvent me) {
    int movingy = me.getY();
    vertical.setValue(movingy);
    repaint();
  }
  // Necessary for MouseMotionListener
  public void mouseMoved(MouseEvent me) {
  }
  // Display current value scroll bars.
  public void paint(Graphics g) {
    message = "Vertical: " + vertical.getValue();
    g.drawString(message, 15, 150);
  }
}
```

The above code shows how to create vertical scrollbars on an applet.

NOTES

Output of the Program:



Check Your Progress

7. State the advantages and disadvantages of a Java applet.
8. Define the features of AWT Packages.
9. What is component class?
10. Define the use of `setTitle()` and `Show()` methods in the applet programming.
11. Layout the various types of constructors of the `TextField` class.
12. What is the use of the constructor `public TextArea()`?
13. Why are scrollbar used for?

12.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. An **applet** is a dynamic and interactive program that can run inside a web page displayed by a Java-capable browser such as a HotJava Browser or an Internet Explorer browser, which are World Wide Web browsers used to view web pages. An applet is a class present in a java.applet package.

NOTES

2. A special HTML tag is embedded in an applet to make it run on the web browser**.** The **appletviewer** application, present in the **jdk**, is used to run and check the applets.
3. The `<APPLET>` tag discussed above includes the following:
   - Name of the Applet
   - Width of the Applet (in pixels)
   - Height of the Applet (in pixels)
4. The following program code creates an HTML web page that embeds an applet:

```
<HTML>
    <HEAD>
       <TITLE>First Applet Program</TITLE>
    </HEAD>
    <BODY>
       <APPLET CODE = "FirstApplet.class" HEIGHT = 150
    WIDTH = 300>
          </APPLET>
       </BODY>
    </HTML>
```

   The above code creates an HTML file in which the <APPLET> tag is used to embed the applet. The attribute of the <APPLET> tag, HEIGHT and WIDTH, sets the dimension of the applet window.
5. **`CODEBASE:`** Specifies the base URL of the applet class file. The base URL is the complete path of the applet where it is stored. This is an optional attribute that searches the executable file of an applet, class file, from the specified URL.
6. Java provides four methods to change the state of an applet. These methods are as follows:
   - `init()`
   - `start()`
   - `stop()`
   - `destroy()`

NOTES

7. **Advantages of a Java Applet**

   A Java **applet** has the following advantages:

   (a) The applet can work properly on all versions of installed Java, excluding the latest plug-in version.

   (b) Almost all web browsers support the applet.

   (c) A user can permit it to have full access to the machine on which it is running.

   (d) It can improve with use, which means that after a first applet is run, the JVM is already running and starts quickly, benefitting the regular Java users. However, the JVM will need to restart each time the browser starts a fresh.

   (e) In terms of its speed of execution, it is slower than C++ codes, but faster than JavaScript.

   Disadvantages of a Java Applet

   A Java applet has the following disadvantages:

   (a) Sometimes, the Java plug-in is required, but it is not available on every web browser by default.

   (b) The process of loading an applet is very slow.

   It can now be concluded that without an applet, Java cannot get the honour of a web-enabled language. The AWT package present in Java is an important package. It is needed to develop a sophisticated applet.

8. The features of the AWT package are:
   - It provides graphics and imaging tools that enable you to develop GUI.
   - It provides layout managers that enable an end user to develop window layouts.
   - It supports event handling.

9. Component class defines the various methods that are used to handle events, define the layout of a component, provide the content of the component and repaint the window during the execution of a program.

10. The `setTitle()` method changes the title of a frame by accepting the new title as an argument passed to it. The `Show()` method is used to make the frame window visible and active on the screen. The frame inherits the color class from its super classes. The color class sets the colour of the frame.

NOTES

11. The various constructors of the textField class are:
    - `public TextField()`
    - `public TextField(int num)`
    - `public TextField(String txt)`
    - `public TextField(String txt, int num)`
12. The constructor, `public TextArea()` is the empty constructor. It creates a text area box with the default settings for width and height.
13. Scrollbars allow end users to select multiple options from the list continuously. Scrollbars are used when all the options in the list box cannot be displayed on the screen at the same time. The scrollbars allow the end user to select the options that are hidden. To select the hidden options, you first need to move the scrollbars to display the options on the screen and then select the options. Scrollbars can be horizontal or vertical.

12.8 SUMMARY

- Applets are executed in the web browser and have made Java a web-enabled language. Applet codes also embed the HTML tags together. This is the package that has made Java a language for distributed application.
- An applet is executed in the browser. An applet is a class file that displays graphics application in the web browser and one can embed applet codes in the web pages by HTML tags.
- An applet is a Java byte code embedded in an HTML page. The program structure of applets differs from the other Java applications.
- An applet is a dynamic and interactive program that can run inside a web page displayed by a Java-capable browser, such as a HotJava Browser or an Internet Explorer browser, which are World Wide Web browsers used to view web pages. An applet is a class present in a java.applet package.
- A special HTML tag is embedded in an applet to make it run on the web browser. An applet has added advantages, such as frame, event-handing facility, graphics context and surrounding user interfaces.
- The body section of the HTML file contains a pair of `<APPLET...>` and `</APPLET>` tags, which allows one to provide the name of the applet and helps the browser recognize the space required for the applet.
- The applet class is a member of the Java Application Programming Interface (API) package, java.applet. The applet class is used for creating a Java program that displays an applet.

NOTES

- Applets are programs based on the Internet that can be used to display text, image or animation.
- To run the applet, the HTML file that embeds an applet on the web page has to be created using the <APPLET> tag.
- The <APPLET> tag enables you to pass the applet parameter in the applet using the <PARAM> tag. The <PARAM> tag has two attributes, NAME and VALUE, which are used to set the name and values of the parameter.
- CODEBASE specifies the base URL of the applet class file. The base URL is the complete path of the applet where it is stored. This is an optional attribute that searches the executable file of an applet, class file, from the specified URL.
- CODE specifies the name of the file that contains a class file. This is a compulsory attribute of the <APPLET> tag that is relative to the URL of the HTML file.
- ALT specifies the tool tip text of an applet. This is also an optional attribute that is displayed to provide information about the applet whether the applet is running or not on the web browser.
- NAME specifies the name of an applet. This is an optional attribute of the <APPLET> tag. The name of an applet is obtained by the `getApplet()` method.
- Each applet class inherits the properties and methods of the class, Applet. An applet is loaded on the web browser or the appletviewer tool of Java. An Applet may change its current state when a specific method is called by AWT.
- The `destroy()` method is called when an applet is destroyed.
- There are four states in the life cycle of an applet: new born, running, idle and dead. The newborn state consists of a newly loaded applet that is initializing its resources.
- An applet uses the classes and methods of AWT to perform its input and output operations. To display the output in the applet, one uses the `drawstring()` method present in the graphics class.
- AppletContext is an interface that is implemented by an object that represents the environment of the applet. To use it, one has to first create an object of `AppletContext`, using the `getAppletContext()` method.
- Java Development Kit (JDK) provides a package called the Abstract Window Toolkit (AWT), which enables you to design user-friendly interfaces.

NOTES

- The AWT package is a collection of classes and methods that allow an end user to design and manage Graphical User Interface (GUI) based applications.
- AWT is used to support applet windows and it also helps in creating independent windows, such as frame windows or panel windows that run in a GUI environment.
- AWT is Java's largest package and can be included in a Java program by giving the java.awt.* statement. It is a part of Java Foundation Classes (JFC). AWT has 63 classes and 14 interfaces used for creating user interfaces.
- Scrollbars allow end users to select multiple options from the list continuously. Scrollbars are used when all the options in the list box cannot be displayed on the screen at the same time.
- The scrollbars allow the end user to select the options that are hidden. To select the hidden options, you first need to move the scrollbars to display the options on the screen and then select the options. Scrollbars can be horizontal or vertical.

12.9 KEY WORDS

- **Applet:** An applet is a dynamic and interactive program that can run inside a web page displayed by a Java-capable browser such as a HotJava Browser or an Internet Explorer browser, which are World Wide Web browsers used to view web pages. An applet is a class present in a java.applet package.
- **Applet class:** The applet class is a member of the Java Application Programming Interface (API) package, java.applet, the applet class is used for creating a Java program that displays an applet.
- **CODE:** Specifies the name of the file that contains a class file. This is a compulsory attribute of the <APPLET> tag that is relative to the URL of the HTML file.
- **ALT:** Specifies the tool tip text of an applet. This is also an optional attribute that is displayed to provide information about the applet whether the applet is running or not on the web browser.
- **NAME:** Specifies the name of an applet. This is an optional attribute of the <APPLET> tag. The name of an applet is obtained by the `getApplet()` method.
- **WIDTH:** Specifies the width of the applet display area in pixels.
- **HEIGHT:** Specifies the height of the applet display area in pixels.

- **ALIGN:** Specifies the alignment of the applet. The values of the ALIGN attribute are LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE and ABSBOTTOM.
- **VSPACE:** Sets the vertical space of the applet, in pixels, on each side of the applet.
- **HSPACE:** Sets the horizontal space of the applet, in pixels, on each side of the applet.

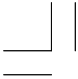
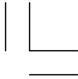
12.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is the importance of applet programming in Java?
2. Why are applets considered to be highly secure programs?
3. How applets differ from applications?
4. How is an applet added to an HTML file?
5. What is an applet tag?
6. Write a short note on AppleContext.
7. What are the two types of constructors that can be used to create frames?

Long-Answer Questions

1. Briefly discuss the significance of Java applet programming.
2. Write applets to building applet code with the help of Java program.
3. Explain the various stages of applet life cycle.
4. Create an executable Java applet for designing a web page.
5. Enumerate the nine attributes of the <APPLET> tag.
6. Write a Java applet program for adding applet to HTML file.
7. Write a Java applet program for running the applet and passing parameters to applets.
8. Explain the various classes used in Java for creating AWT controls.
9. Write a Java applet program for displaying numerical values and getting input from the user.
10. Explain the various applet class methods used for displaying text and images, playing audio files and responding when you interact with the applet.

12.11 FURTHER READINGS

NOTES

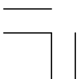
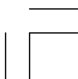
Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

NOTES

BLOCK - V
MANAGING INPUT/OUTPUT FILES IN JAVA

UNIT 13 INTRODUCTION TO STREAMS

Structure

13.0 INTRODUCTION

A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result. The Java stream is not a data structure instead it takes input from the collections, arrays or I/O channels. Streams do not change the original data structure, they only provide the result as per the pipelined methods. Fundamentally, stream does not store elements. It simply conveys elements from a source, such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.

In this unit, you will study about the concept of streams in Java, stream classes, byte stream classes and character stream classes.

13.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the concept of streams in Java
- Discuss about stream classes in Java
- Use byte stream classes in Java programs
- Understand the importance of character stream classes in Java programs

13.2 I/O BASICS

NOTES

Most real-life applications require large amount of input and output data to be handled that is difficult to manage using commonly used console input/output (I/O) devices, such as a keyboard and a screen. Moreover, the output obtained using console I/O operation is not permanent and is lost as soon as the program terminates. This necessitates the requirement of some devices, such as hard disk, magnetic tape or a floppy disk to store voluminous data permanently. The data is stored in these devices in the form of files (also known as **disk files)** and is called **persistent data**.

It is significant for data to be persistent, i.e., it should survive when the program is finished.

Whatever values stored in variables do not persist, or survive, when the program is finished. Instead, the data is lost because the data is stored in RAM (random access memory), which is cleared when the program (or the computer) stops running.

Most programs are capable of saving data to a file on the computer's hard drive or other storage medium so that data later can be retrieved when needed. That data persists after the termination of the program or even after the computer is turned off.

A file is a collection of data, and is located on persistent storage, such as a hard drive, a CD-ROM, or other storage device.

A file can be read or written in the form of bytes or characters using byte stream classes and character stream classes, respectively. The objects of classes can also be read from and written to the secondary memory (files). The process of reading and writing objects is known as **object serialization**. Serialization refers to the conversion of an object to a storable bit sequence. Serialization primary deals with transforming a binary object into an XML (or other string) representation so that it can be stored in a database/file or sent across a network in a web service call. It applies to objects such as a String. A serialized object is a standard Java object, but it must implement the `java.io.Serializable` interface to be used with object serialization. The serializable interface does not have any methods, and is used to point to the object that must be serialized. Object serialization needs an instance of `ObjectOutputStream`, which is a subclass of `FilterOutputStream`. `ObjectOutputStream` wraps itself around another output stream to use the output functionality.

13.3 STREAMS AND STREAM CLASSES

Java manages all input and output in the form of streams. A **stream** refers to a channel through which data flows from the source to the destination. This data is in the form of sequence of bytes or characters.

Java streams can be broadly categorized into two types, namely, input stream and output stream. The streams which help to read data from various sources, namely, keyboard, mouse, files, storage devices, etc., in order to supply it to the program are known as **input streams**. The streams which receive data from the program and directly write it to the physical devices or other programs are called **output streams**. For instance, to bring the data from an input device into the program, the program opens an input stream on the input device and reads the data in a serial manner. Conversely, to write data from the program to an output device, the program opens an output stream to the output device and writes data to it serially (see Figure 13.1).

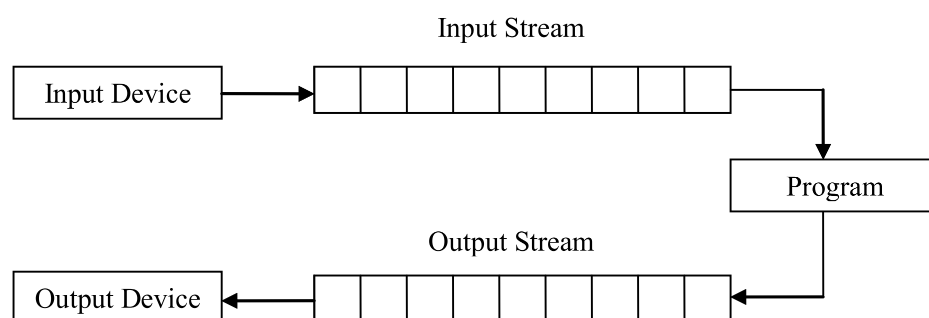


Fig. 13.1 *Java Streams*

Java supports input/output streams through a hierarchy of classes defined in the `java.io` package. On the basis of the type of data on which these classes operate, they can be categorized into two groups, byte stream classes and character stream classes (see Figure 13.2).

- **Byte Stream Classes:** These classes support input and output operations on bytes (8-bit bytes). For example, while reading from or writing data to a binary file, byte stream classes are used.
- **Character Stream Classes:** These classes perform input and output operations on characters (16-bit Unicode). For example, while reading from or writing data to a text file, character stream classes are used.

NOTES

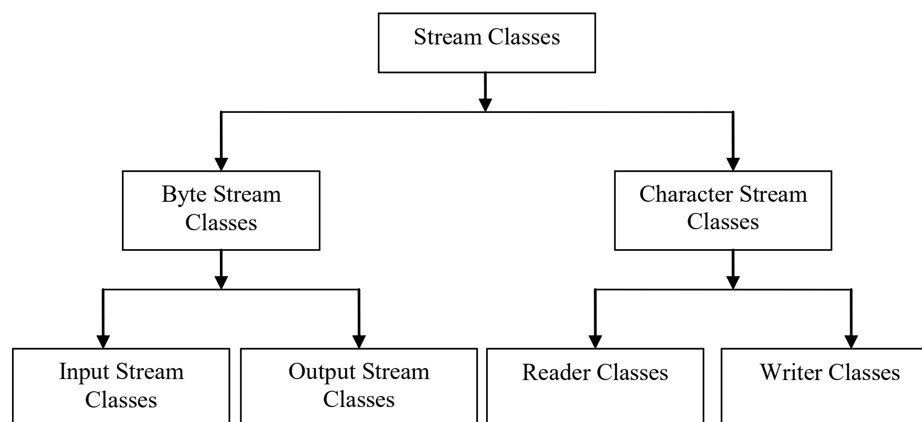


Fig. 13.2 *Classification of Stream Classes*

13.3.1 Byte Stream Classes

Byte stream classes are used to perform reading and writing of 8-bit bytes. Streams being unidirectional in nature can transfer bytes in one direction only, i.e., either reading data from the source into a program or writing data from a program to the destination. Therefore, Java further divides byte stream classes into two classes, namely, `InputStream` class and `OutputStream` class. The subclasses of `InputStream` class contain methods to support input and the subclasses of `OutputStream` class contain output related methods.

Input Stream Classes

Java's input stream classes are used to read 8-bit bytes from the stream. The `InputStream` class is the superclass for all byte-oriented input stream classes. All the methods of this class throw an `IOException`. Being an abstract class, the `InputStream` class cannot be instantiated hence, its subclasses are used. Some of these are listed in Table 13.1.

Table 13.1 `InputStream` *Classes*

Class	Description
`BufferedInputStream`	contains methods to read bytes from the buffer (memory area)
`ByteArrayInputStream`	contains methods to read bytes from a byte array
`DataInputStream`	contains methods to read Java primitive data types
`FileInputStream`	contains methods to read bytes from a file
`FilterInputStream`	contains methods to read bytes from other input streams which it uses as its basic source of data
`ObjectInputStream`	contains methods to read objects
`PipedInputStream`	contains methods to read from a piped output stream. A piped input stream must be connected to a piped output stream
`SequenceInputStream`	contains methods to concatenate multiple input streams and then read from the combined stream

The `InputStream` class defines various methods to perform reading operations on data of an input stream. Some of these methods along with their description are listed in Table 13.2.

NOTES

Table 13.2 `InputStream` *Class Methods*

Method	Description
`int read()`	returns the integral representation of the next available byte of input. It returns -1 when end of file is encountered
`int read(byte buffer[])`	attempts to read `buffer.length` bytes into the `buffer` and returns the total number of bytes successfully read. It returns -1 when end of file is encountered
`int read(byte buffer[], int loc, int nBytes)`	attempts to read '`nBytes`' bytes into the `buffer` starting at `buffer[loc]` and returns the total number of bytes successfully read. It returns -1 when end of file is encountered
`int available()`	returns the number of bytes of the input available for reading
`void mark(int nBytes)`	marks the current position in the input stream until '`nBytes`' bytes are read
`void reset()`	resets the input pointer to the previously set mark
`long skip(long nBytes)`	skips '`nBytes`' bytes of the input stream and returns the number of actually skipped bytes
`void close()`	closes the input source. If an attempt is made to read even after closing the stream then it generates `IOException`

Using `ByteArrayInputStream` Class

The `ByteArrayInputStream` class opens an input stream to read bytes from a byte array. It contains an internal buffer which holds bytes that are read from the stream. It should be noted that closing the stream does not have any consequences. This implies that methods of this class can be invoked even after closing the stream without generating any `IOException`.

The `ByteArrayInputStream` object can be created using one of the following constructors:

```
ByteArrayInputStream(byte[] buffer) //first
ByteArrayInputStream(byte[] buffer, int loc, int nBytes)
//second
```

The first constructor creates a `ByteArrayInputStream` which uses a byte array `buffer` as its input source. The second constructor creates a `ByteArrayInputStream` which uses a subset of byte array `buffer` as its input source. The reading begins from the index specified by `loc` and continues until `nBytes` are read.

NOTES

Program 1: A program to demonstrate the use of `ByteArrayInputStream` class.

```
import java.io.*;
class ByteArrayInputStreamExample
{
    public static void main(String args[]) throws
IOException
    {
        byte b[]="this is my first program".getBytes();

        ByteArrayInputStream inp =new
ByteArrayInputStream(b);
        int n=inp.available();
        System.out.println("Number of available bytes: "+n);
        long s=inp.skip(11);    //skipping 11 bytes
        System.out.println("Number of skipped bytes: "+s);
        int i;
        System.out.print("String after skipping s bytes:
");
        while((i=inp.read())!= -1)
        {
            System.out.print((char)i);
        }
inp.reset(); /*reset the pointer to the beginning of the
stream*/
        System.out.println();   //new line
        int j;
        System.out.print("String in uppercase: ");
        while((j=inp.read())!= -1)
        {
System.out.print(Character.toUpperCase((char) j));
        }
    }
}
```

Output of the program:

```
Number of available bytes: 24
Number of skipped bytes: 11
String after skipping s bytes: first program
String in uppercase: THIS IS MY FIRST PROGRAM
```

In this program, the getBytes() method is used to convert a string into bytes. The use of available() and skip() methods is demonstrated here. Once the entire stream is read, the reset() method is invoked to set the pointer at the start of the stream.

Output Stream Classes

Java's output stream classes are used to write 8-bit bytes to a stream. The OutputStream class is the superclass for all byte-oriented output stream classes. All the methods of this class throw an IOException. Being an abstract class, the OutputStream class cannot be instantiated hence, its subclasses are used. Some of these are listed in Table 13.3.

Table 13.3 OutputStream *Classes*

Class	Description
BufferedOutputStream	contains methods to write bytes into the buffer
ByteArrayOutputStream	contains methods to write bytes into a byte array
DataOutputStream	contains methods to write Java primitive data types
FileOutputStream	contains methods to write bytes to a file
FilterOutputStream	contains methods to write to other output streams
ObjectOutputStream	contains methods to write objects
PipedOutputStream	contains methods to write to a piped input stream
PrintStream	contains methods to print Java primitive data types

The OutputStream class defines methods to perform writing operations. These methods are discussed in Table 13.4.

Table 13.4 OutputStream *Class Methods*

Method	Description
void write(int i)	writes a single byte to the output stream
void write(byte buffer[])	writes an array of bytes to the output stream
void write(bytes buffer[], int loc, int nBytes)	writes 'nBytes' bytes to the output stream from the buffer b starting at buffer[loc]
void flush()	flushes the output stream and writes the waiting buffered output bytes
void close()	closes the output stream. If an attempt is made to write even after closing the stream then it generates IOException

Using ByteArrayOutputStream Class

The ByteArrayOutputStream class, an output counterpart of the ByteArrayInputStream, writes streams of bytes to the buffer. Similar to ByteArrayInputStream, closing this stream has no effect. That is, methods of this class can be invoked even after closing the stream without generating any IOException.

NOTES

The `ByteArrayOutputStream` object can be created using one of the following constructors.

```
ByteArrayOutputStream()   //first
ByteArrayOutputStream(int nBytes)   //second
```

The first constructor creates a buffer of 32 bytes. The second constructor creates a buffer of size equal to `nBytes`. The size of the buffer increases as bytes are written to it.

Program 2: A program to demonstrate the use of `ByteArrayOutputStream` class.

```
import java.io.*;
class ByteArrayOutputStreamExample
{
    public static void main(String args[]) throws
IOException
    {
       ByteArrayOutputStream out=new
ByteArrayOutputStream();
       byte b[]="Today is a bright sunny day".getBytes();
       out.write(b);
System.out.println(out.toString());/*converting byte
              array to String*/
       out.close(); //closing the stream
    }
}
```

Output of the program:

```
Today is a bright sunny day
```

In this program, the `getBytes()` method is used to convert a string into bytes. Once the entire stream is written, the `toString()` method is invoked to convert the contents (bytes) of the buffer into a string.

13.3.2 Character Stream Classes

One of the limitations of byte stream classes is that it can handle only 8-bit bytes and cannot work directly with Unicode characters. To overcome this limitation, character stream classes have been introduced in `java.io` package to match the byte stream classes. The character stream classes support 16-bit Unicode characters, performing operations on characters, character arrays, or strings, reading or writing buffer at a time. Character stream classes are divided into two stream classes, namely `Reader` class and `Writer` class.

NOTES

Reader Classes

Reader classes are used to read 16-bit Unicode characters from the input stream. The `Reader` class is the superclass for all character-oriented input stream classes. All the methods of this class throw an `IOException`. Being an abstract class, the `Reader` class cannot be instantiated hence its subclasses are used. Some of these are listed in Table 13.5.

Table 13.5 *Reader Classes*

Class	Description
`BufferedReader`	contains methods to read characters from the buffer
`CharArrayReader`	contains methods to read characters from a character array
`FileReader`	contains methods to read from a file
`FilterReader`	contains methods to read from underlying character-input stream
`InputStreamReader`	contains methods to convert bytes to characters
`PipedReader`	contains methods to read from the connected piped output stream
`StringReader`	contains methods to read from a string

The `Reader` class defines various methods to perform reading operations on data of an input stream. Some of these methods along with their description are listed in Table 13.6.

Table 13.6 `Reader` *Class Methods*

Method	Description
`int read()`	returns the integral representation of the next available character of input. It returns -1 when end of file is encountered
`int read(char buffer[])`	attempts to read `c.length` characters into the `buffer` and returns the total number of characters successfully read. It returns -1 when end of file is encountered
`int read(char buffer[], int loc, int nChars)`	attempts to read '`nChars`' characters into the `buffer` starting at `buffer[loc]` and returns the total number of characters successfully read. It returns -1 when end of file is encountered
`void mark(int nChars)`	marks the current position in the input stream until '`nChars`' characters are read
`void reset()`	resets the input pointer to the previously set mark
`long skip(long nChars)`	skips '`nChars`' characters of the input stream and returns the number of actually skipped characters
`boolean ready()`	returns `true` if the next request of the input will not have to wait, else it returns `false`
`void close()`	closes the input source. If an attempt is made to read even after closing the stream then it generates `IOException`

NOTES

Using **BufferedReader** Class

The `BufferedReader` class creates a buffered character stream between the input device and the program. It defines the methods to read the data from the buffer in the form of characters.

The `BufferedReader` object can be created using one of the following two constructors:

```
BufferedReader(Reader inpStream)        //first
BufferedReader(Reader inpStream, int nChars) //second
```

The first constructor creates a buffered character stream of a default buffer size. The second constructor creates a buffered character stream that uses an input buffer of size `nChars`.

Besides the methods provided by the `Reader` class, the `BufferedReader` class contains some other methods which are discussed as follows:

- `String readLine()`: It reads a line of the input text.
- `Boolean ready()`: It checks whether or not the stream is ready to be read.

Program 3: A program to demonstrate the use of `BufferedReader` class.

```
import java.io.*;
class BufferedReaderExample
{
public static void main(String args[]) throws IOException
{
String s="This program is skipping first character of
each word in the string";
StringReader sr=new StringReader(s);
System.out.println("The input string is: "+s);
/*Creating an instance of BufferedReader using
StringReader*/
BufferedReader br=new BufferedReader(sr);
//Reading from the underlying StringReader
System.out.print ("The output string is: ");
br.skip(1); /*skipping the first character of the  first
word of the input string*/
//reading the remaining string
```

```
int i=0;
while((i=br.read())!=-1)
{
System.out.print((char)i);
if((char)i==' ') //checking for white spaces
{
br.skip(1); /*skipping first character of  each word*/
          }
       }
    }
}
```

NOTES

Output of the program:

```
The input string is: This program is skipping first character
of each word in the string
The output string is: his rogram s kipping irst haracter f ach
ord n he tring
```

In this program, the `skip()` method is used to skip the first character of each word of the input string.

Writer Classes

Writer classes are used to write 16-bit Unicode characters onto an output stream. The `Writer` class is the superclass for all character-oriented output stream classes. All the methods of this class throw an `IOException`. Being an abstract class, the `Writer` class cannot be instantiated hence, its subclasses are used. Some of these are listed in Table 13.7.

Table 13.7 *Writer Classes*

Class	Description
`BufferedWriter`	contains methods to write characters to a buffer
`FileWriter`	contains methods to write to a file
`FilterWriter`	contains methods to write characters to underlying output stream
`CharArrayWriter`	contains methods to write characters to a character array
`OutputStreamWriter`	contains methods to convert from bytes to characters
`PipedWriter`	contains methods to write to the connected piped input stream
`StringWriter`	contains methods to write to a string

The `Writer` class defines various methods to perform writing operations on output stream. These methods along with their description are listed in Table 13.8.

NOTES

Table 13.8 `Writer` *Class Methods*

Method	Description
`void write()`	writes data to the output stream
`void write(int i)`	writes a single character to the output stream
`void write(char buffer[])`	writes an array of characters to the output stream
`void write(char buffer[], int loc, int nChars)`	writes 'n' characters from the `buffer` starting at `buffer[loc]` to the output stream
`void close()`	closes the output stream. If an attempt is made to perform writing operation even after closing the stream then it generates `IOException`
`void flush()`	flushes the output stream and writes the waiting buffered output characters

Using `BufferedWriter` class

The `BufferedWriter` class, an output counterpart of the `BufferedReader`, is used to write characters onto the character-output stream.

The `BufferedWriter` object can be created using one of the following two constructors.

```
BufferedWriter(Writer outStream)        //first
BufferedWriter(Writer outStream, int nChars) //second
```

The first constructor creates a buffered character stream of default buffer size. The second constructor creates a buffered character stream that uses an input buffer of size `nChars`.

Besides the methods provided by the `Writer` class, the `BufferedWriter` class contains the `newLine()` method which writes a new line.

Program 4: A program to demonstrate the use of `BufferedWriter` class.

```
import java.io.*;
class BufferedWriterExample
{
    public static void main(String args[]) throws
IOException
    {
        String s="This is an example of BufferedWriter";
        StringWriter sw=new StringWriter();
        //creating an instance of BufferedWriter class
```

NOTES

```
        BufferedWriter bw=new BufferedWriter(sw);
        bw.write(s,0,5);
        bw.newLine();
bw.write(s,5,s.length()-5);/*writes the substring starting
from index 5*/
        bw.flush();    //flushes the stream
        System.out.println(sw.getBuffer());
        sw.close();    //closing the stream
        bw.close();    //closing the stream
    }
}
```

Output of the program:

```
This
is an example of BufferedWriter
```

In this illustration, the `BufferdWriter` class is used to write data to character-output stream (instance of `StringWriter` class). The `getBuffer()` method of the `StringWriter` class returns a buffer of `String` type.

13.3.3 The `PrintStream` Class

When a program is executed, the input is read from the various sources and the output is sent to different destinations. Generally, a keyboard and monitor screen are used as standard input and output device, respectively. The data fed by the user is supplied to the program using input streams and the output of the program is supplied to the output device using output streams. This output is displayed to the user using Java `PrintStream` class. So far the two methods, `print()` and `println()`, that have been used for displaying the primitive data type, object, etc., on an output device are defined by the `PrintStream` class. This is a byte stream class which is derived from the `OutputStream` class. Unlike other output streams, the `PrintStream` class does not throw an `IOException` even if an exceptional event occurs.

The `PrintStream` class can be connected to the underlying output stream, such as `FileOutputStream`, `ByteArrayOutputStream`, `BufferedOutputStream`, etc.

`PrintStream` class defines the following type of constructor:

```
PrintStream(OutputStream os)
```

This constructor creates a print stream and connects it to the output stream os.

NOTES

In addition to print() and println() methods, the PrintStream class defines some other methods which are listed in Table 13.9.

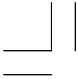
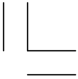
Table 13.9 PrintStream *Class Methods*

Method	Description
PrintStream append(char c)	appends the character c to the output stream
PrintStream append(CharSequence cs)	appends the character sequence c to the output stream
boolean checkError()	checks the error state of the stream
protected void setError()	sets the error state to true

13.3.4 Predefined Streams

The package java.lang is automatically imported by all Java programs. This package includes a class System, which holds a large number of predefined static methods and variables. This class contains three predefined stream variables including in, out and err. These stream variables are declared as static and public and are initialized automatically with the start up of JVM. They can be used by any part of the program easily without referring to a specific System object.

- **System.out:** It is an object of class type PrintStream.. It refers to the standard output stream and is generally used to write text output to the console.
- **System.in:** It is an object of class type InputStream. It refers to the standard input stream and is generally used to read keyboard input in console programs.
- **System.err:** It is also an object of class type PrintStream. It refers to the standard error stream and works in a manner similar to the System.out except that it displays error messages.

Check Your Progress

1. Differentiate between byte stream classes and character stream classes.
2. Explain about the object serialization.
3. Differentiate between input streams and output streams.
4. What do you mean by byte stream classes?
5. Why is `void mark(int nBytes)` used in Java?
6. State the different types of methods used to perform reading operations on data of an input stream.
7. What are the uses of writer class in stream?
8. How can the `PrintStream` class connect to the output stream?
9. What are the functions of `System.in` object?

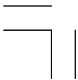
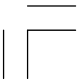
NOTES

13.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Byte stream classes support input and output operations on bytes (8-bit bytes), whereas character stream classes perform input and output operations on characters (16-bit Unicode).

   CYP

2. The process of reading and writing objects is known as **object serialization**. Serialization refers to the conversion of an object to a storable bit sequence. Serialization primary deals with transforming a binary object into an XML (or other string) representation so that it can be stored in a database/file or sent across a network in a web service call. It applies to objects such as a String. A serialized object is a standard Java object, but it must implement the `java.io.Serializable` interface to be used with object serialization. The serializable interface does not have any methods, and is used to point to the object that must be serialized. Object serialization needs an instance of `ObjectOutputStream`, which is a subclass of `FilterOutputStream`. `ObjectOutputStream` wraps itself around another output stream to use the output functionality.
3. The streams which help to read data from various sources, namely, keyboard, mouse, files, storage devices, etc., in order to supply it to the program are known as **input streams**. The streams which receive data from the program and directly write it to the physical devices or other programs are called **output streams**.

NOTES

4. These classes support input and output operations on bytes (8-bit bytes).
5. `int read(byte buffer[], int loc, int nBytes)` attempts to read '`nBytes`' bytes into the `buffer` starting at `buffer[loc]` and returns the total number of bytes successfully read. It returns -1 when end of file is encountered.
6. The `Reader` class defines various methods to perform reading operations on data of an input stream. Some of these methods along with their description are listed in Table 13.6.

Table 13.6 `Reader` *Class Methods*

Method	Description
`int read()`	returns the integral representation of the next available character of input. It returns -1 when end of file is encountered
`int read(char buffer[])`	attempts to read `c.length` characters into the `buffer` and returns the total number of characters successfully read. It returns -1 when end of file is encountered
`int read(char buffer[], int loc, int nChars)`	attempts to read '`nChars`' characters into the `buffer` starting at `buffer[loc]` and returns the total number of characters successfully read. It returns -1 when end of file is encountered
`void mark(int nChars)`	marks the current position in the input stream until '`nChars`' characters are read
`void reset()`	resets the input pointer to the previously set mark
`long skip(long nChars)`	skips '`nChars`' characters of the input stream and returns the number of actually skipped characters
`boolean ready()`	returns `true` if the next request of the input will not have to wait, else it returns `false`
`void close()`	closes the input source. If an attempt is made to read even after closing the stream then it generates `IOException`

7. Writer classes are used to write 16-bit Unicode characters onto an output stream. The `Writer` class is the superclass for all character-oriented output stream classes. All the methods of this class throw an `IOException`. Being an abstract class, the `Writer` class cannot be instantiated hence, its subclasses are used. The `Writer` class defines various methods to perform writing operations on output stream.
8. The `PrintStream` class can be connected to the underlying output stream, such as `FileOutputStream`, `ByteArrayOutputStream`, `BufferedOutputStream`, etc.

   `PrintStream` class defines the following type of constructor:

   `PrintStream(OutputStream os)`

   This constructor creates a print stream and connects it to the output stream `os`.

9. **System.in:** It is an object of class type InputStream. It refers to the standard input stream and is generally used to read keyboard input in console programs.

13.5 SUMMARY

NOTES

- Most real-life applications require large amount of input and output data to be handled that is difficult to manage using commonly used console input/ output (I/O) devices, such as a keyboard and a screen.
- The objects of classes can also be read from and written to the secondary memory (files). The process of reading and writing objects is known as **object serialization**.
- Java streams can be broadly categorized into two types, namely, input stream and output stream. The streams which help to read data from various sources, namely, keyboard, mouse, files, storage devices, etc., in order to supply it to the program are known as **input streams**. The streams which receive data from the program and directly write it to the physical devices or other programs are called **output streams**.
- **Byte Stream Classes:** These classes support input and output operations on bytes (8-bit bytes). For example, while reading from or writing data to a binary file, byte stream classes are used.
- These classes perform input and output operations on characters (16-bit Unicode).
- Java's input stream classes are used to read 8-bit bytes from the stream. The InputStream class is the superclass for all byte-oriented input stream classes.
- The ByteArrayInputStream class opens an input stream to read bytes from a byte array. It contains an internal buffer which holds bytes that are read from the stream.
- Java's output stream classes are used to write 8-bit bytes to a stream. The OutputStream class is the superclass for all byte-oriented output stream classes.
- One of the limitations of byte stream classes is that it can handle only 8-bit bytes and cannot work directly with Unicode characters. To overcome this limitation, character stream classes have been introduced in java.io package to match the byte stream classes.
- Reader classes are used to read 16-bit Unicode characters from the input stream. The Reader class is the superclass for all character-oriented input stream classes.

NOTES

- The first constructor creates a buffered character stream of default buffer size. The second constructor creates a buffered character stream that uses an input buffer of size `nChars`.
- Generally, a keyboard and monitor screen are used as standard input and output device, respectively. The data fed by the user is supplied to the program using input streams and the output of the program is supplied to the output device using output streams. This output is displayed to the user using Java `PrintStream` class. So far the two methods, `print()` and `println()`, that have been used for displaying the primitive data type, object, etc., on an output device are defined by the `PrintStream` class. This is a byte stream class which is derived from the `OutputStream` class. Unlike other output streams, the `PrintStream` class does not throw an `IOException` even if an exceptional event occurs.

  The `PrintStream` class can be connected to the underlying output stream, such as `FileOutputStream`, `ByteArrayOutputStream`, `BufferedOutputStream`, etc.

  `PrintStream` class defines the following type of constructor:

  ```
  PrintStream(OutputStream os)
  ```

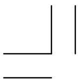
  This constructor creates a print stream and connects it to the output stream `os`.
- Predefined Streams

  The package `java.lang` is automatically imported by all Java programs. This package includes a class `System`, which holds a large number of predefined static methods and variables. This class contains three predefined stream variables including `in`, `out` and `err`. These stream variables are declared as `static` and `public` and are initialized automatically with the start up of JVM. They can be used by any part of the program easily without referring to a specific `System` object.

  o **`System.out`:** It is an object of class type `PrintStream`.. It refers to the standard output stream and is generally used to write text output to the console.

  o **`System.in`:** It is an object of class type `InputStream`. It refers to the standard input stream and is generally used to read keyboard input in console programs.

  o **`System.err`:** It is also an object of class type `PrintStream`. It refers to the standard error stream and works in a manner similar to the `System.out` except that it displays error messages.




NOTES

13.6 KEY WORDS

- **Character Stream Classes:** These classes perform input and output operations on characters (16-bit Unicode). For example, while reading from or writing data to a text file, character stream classes are used.
- **`String readLine():`** It reads a line of the input text.
- **`Boolean ready():`** It checks whether or not the stream is ready to be read.
- **`System.out:`** It is an object of class type `PrintStream`.. It refers to the standard output stream and is generally used to write text output to the console.
- **`System.in:`** It is an object of class type `InputStream`. It refers to the standard input stream and is generally used to read keyboard input in console programs.
- **`System.err:`** It is also an object of class type `PrintStream`. It refers to the standard error stream and works in a manner similar to the `System.out` except that it displays error messages.

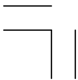
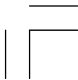
13.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

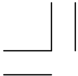
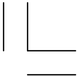
Short Answer Questions

1. Define stream.
2. Differentiate between byte stream classes and character stream classes.
3. What are the `inputstream` classes used in the stream?
4. Explain output stream classes.
5. State the features of `void close()` in reader class methods.
6. Describe the predefined streams.

Long Answer Questions

1. Explain the Java streams along with examples.
2. Discuss all the `InputStream` class methods.
3. Classify the different types of stream classes.
4. Write a program to illustrate the use of `ByteArrayInputStream` method.
5. Elaborate the various types of stream variables.

13.8 FURTHER READINGS

NOTES

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

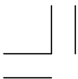
Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.




UNIT 14 I/O CLASSES

NOTES

Structure

14.0 INTRODUCTION

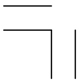
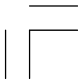
Java I/O (Input and Output) is used to process the input and produce the output. Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations. A file is a collection of related data stored on some storage device. The data stored in a data file is permanent and can easily be accessed as and when required. Moreover, the data stored in a file by one program can be accessed or modified by various programs as per requirements. For reading and writing data to the files, file streams are used. All these streams represent an input source and an output destination. The stream in the java.io package supports many data, such as primitives, object, localized characters, etc.

In this unit, you will study about the concept of I/O classes using stream, reading and writing files, and reading writing bytes in Java.

14.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the concept of I/O classes using stream in Java
- Discuss about reading and writing files in Java
- Understand the importance of reading writing bytes in Java programs

14.2 READING FROM AND WRITING TO CONSOLE

NOTES

Java 1.0 supports console input using byte streams. Since the use of byte streams for reading/writing console input requires using deprecated methods, this approach is not recommended. The more desired approach used for reading input for Java 2 is to use character streams instead of byte streams. Java 1.1 and higher versions of Java provide `InputStreamReader` to convert byte-oriented data into character-oriented data. In order to read data from the console using character stream, `System.in` is used. As `System.in` refers to an object of type `InputStream`, it can be used as a character-based input stream.

To write the data to the console, the `System.out` stream is used which belongs to the `PrintStream` class. The `PrintStream` class defines two methods to write output to console, one is `print()` and the other is `println()`. Both methods perform the same task except that the `println()` method writes the output in a new line while `print()` method writes the output in the same line.

The statement to wrap `InputStreamReader` inside the `BufferedReader` and create a character-based stream connected to the console through `System.in` is as follows:

```
BufferedReader br = new BufferedReader (new
InputStreamReader (System.in));
```

Program 1: A program to demonstrate the use of `System.out` and `System.in`.

```
import java.io.*;
class InputStreamReaderExample
{
    public static void main(String args[])
    {
        try
        {
            int j[]=new int[5];
        double sum=0.0;
            System.out.print("Enter your name: ");
BufferedReader br=new BufferedReader (new
InputStreamReader (System.in));

//reading input from the keyboard
            String name=br.readLine();
System.out.println("Enter marks you scored in five
subjects");
        for (int i=0; i<j.length; i++)
        {
```

```
                System.out.print((i+1)+": ");
                String s=br.readLine();
                j[i]=Integer.parseInt(s);
                sum+=j[i];
            }
    System.out.println(name+" your percentage is: " +sum/
    j.length+"%");
        }
        catch(Exception e){}
    }
}
```

Output of the program:

```
Enter your name: Kevin
Enter marks you scored in five subjects
1: 89
2: 90
3: 65
4: 78
5: 60
Kevin your percentage is: 76.4%
```

14.3 READING AND WRITING FILES

A **file** is a collection of related data stored on some storage device. The data stored in a data file is permanent and can easily be accessed as and when required. Moreover, the data stored in a file by one program can be accessed or modified by various programs as per requirements. For reading and writing data to the files, file streams are used. A file stream refers to the flow of data between the program and the files (see Figure 14.1). Depending on the type of data handled by the file stream, byte stream classes or character stream classes can be used.

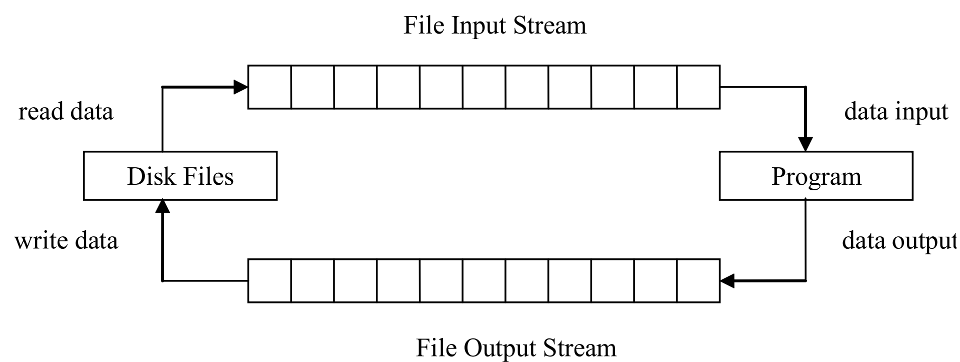


Fig. 14.1 *File Input and Output Stream*

NOTES

Reading and Writing Bytes to a File

Before reading or writing a file, it must be opened first. For this, it is required to create a file stream and then link this stream to the specific file. Java provides `FileInputStream` class to read bytes from a file. The `FileInputStream` class creates an input stream between the file and the program and reads bytes from the file and sends it to the program.

The `FileInputStream` defines following type of constructor:

```
FileInputStream(String FilePath)
```

where,

`FilePath` is the full path describing the location of the file

Program 2: A program to demonstrate the use of `FileInputStream` class to read from a file.

```
import java.io.*;
class FileInputStreamExample
{
public static void main(String[] args) throws IOException
    {
FileInputStream fis=new FileInputStream ("student.txt");
int size=fis.available(); /*determine the size of the
file*/
       System.out.println("Size of the file is: "+ size);
       System.out.println("The contents of the file are:
");
       int i;
       while((i=fis.read())!=-1)
       {
          System.out.print((char)i);
       }
       fis.close();
       System.out.println();
       System.out.println("Reading Complete");
    }
}
```

Output of the program:

```
Size of the file is: 118
The contents of the file are:
Name: Amit Sharma
Roll No.: 1
Overall Percentage: 89%
```

```
Name: Shivani Dutta
Roll No.: 2
Overall Percentage: 92%

Reading Complete
```

NOTES

In this program, the statement `FileInputStream fis = new FileInputStream(student.txt)` creates an input stream and causes the file `student.txt` to be opened for reading. After the creation of the stream object, the `fis.read()` method is used to read the next byte from the file `student.txt`.

Similarly, the `FileOutputStream` class defines methods to write bytes to a file.

The `FileOutputStream` defines following type of constructor.

FileOutputStream(String FilePath)

where,

`FilePath` is the full path describing the location of the file.

Program 3: A program to demonstrate the use of `FileOutputStream` class to write to a file.

```
import java.io.*;
class FileOutputStreamExample
{
    public static void main(String[] args) throws
IOException
    {
String filecontents="These contents will be written to
the file";
       byte b[]=filecontents.getBytes();
FileOutputStream fos=new FileOutputStream ("newfile.txt");

       for(int i=0; i<b.length; i++)
       {
          fos.write(b[i]);
       }
       fos.close();
       System.out.println("Write successful");
    }
}
```

Output of the program:

```
Write successful
```

In this program, when the `FileOutputStream` class' object is created, a file with the specified name is automatically created by the operating system. If a file with the same name already exists in the same directory, the contents of the earlier file get overwritten with the contents of the new file.

NOTES

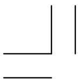
Reading and Writing Characters to a File

The `FileReader` class is used to read characters from the file. The `FileReader` class creates character stream between the file and the program and reads characters from the file and sends it to the program. Similarly, to write characters to a file, the `FileWriter` class is used.

Program 4: A program to demonstrate the use of `FileReader and FileWriter` class for reading from and writing characters to a file.

```
import java.io.*;
class ReadWriteFile
{
   public static void main(String args[]) throws
IOException
   {
      String w="Hello\nHow\nare\nyou";
      FileWriter fw=new FileWriter("data.txt");
      System.out.println("Writing to the file
data.txt...");
      fw.write(w);
      System.out.println("Writing complete");
      fw.close();

      System.out.println();
      FileReader fr=new FileReader("data.txt");
      BufferedReader b=new BufferedReader(fr);
      System.out.println("Reading the file data.txt...");
      while((w=b.readLine())!=null)
      {
         System.out.println(w);
      }
      fr.close();
      System.out.println("Reading ends");
   }
}
```




Output of the program:

```
Writing to the file data.txt...
Writing complete

Reading the file data.txt...
Hello
How
are
you
Reading ends
```

NOTES

In this program, when the `FileWriter` class' object is created, a file with the name `data.txt` is automatically created by the operating system and data is written to it. The object of `FileReader` class is used to read the contents of the same file.

14.4 DATA STREAMS

Data streams support reading/writing of primitive data types (`int`, `float`, `short`, `long`, `double`, `boolean`, `char` and `byte`) as well as `String` objects. These are required when the user wants to work on data other than bytes or characters. Data streams are filtered streams, that is, they filter existing stream so that the primitive data types can be directly read from or written to the stream. Java provides `DataInputStream` class and `DataOutputStream` class to read and write primitive data types, respectively.

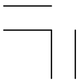
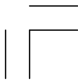
14.4.1 The `DataInputStream` Class

The `DataInputSream` class performs reading of Java primitive data types from the input stream. To create a data input stream, we use the constructor of `DataInputStream` class which is defined as follows:

```
DataInputStream (InputStream inp)
```

where,

`inp` is an existing input stream which is to be filtered, such as `FileInputStream`, `BufferedInputSream`, etc.

NOTES

Some other commonly used methods are listed in Table 14.1.

Table 14.1 *`DataInputStream` Class Methods*

Method	Description
`int readInt()`	reads `integer` type values
`float readFloat()`	reads `float` type values
`char readChar()`	reads `char` type values
`double readDouble()`	reads `double` type values
`long readLong()`	reads `long` type values
`short readShort()`	reads `short` type values
`boolean readBoolean()`	reads `boolean` type values
`byte readByte()`	reads `byte` type values
`String readUTF()`	reads `string` type values

14.4.2 The `DataOutputStream` Class

The `DataOutputStream` class performs writing of Java primitive data types onto the output stream. To create a data output stream, we use the constructor of `DataOutputStream` class which is defined as follows:

```
DataOutputStream(OutputStream out)
```

where,

`out` is an existing output stream which is to be filtered such as `FileOutputStream`, `BufferedInputSream`, etc.

Some other commonly used methods are listed in Table 14.2.

Table 14.2 *`DataOutputStream` Class Methods*

Method	Description
`int writeInt()`	writes `integer` type values
`float writeFloat()`	writes `float` type values
`char writeChar()`	writes `char` type values
`double writeDouble()`	writes `double` type values
`long writeLong()`	writes `long` type values
`short writeShort()`	writes `short` type values
`boolean writeBoolean()`	writes `boolean` type values
`byte writeByte()`	writes `byte` type values
`String writeUTF()`	writes `string` type values

Program 5: A program to demonstrate the use of `DataInputStream` class and `DataOutputStream` class for handling primitive data types.

```
import java.io.*;
class DataStreamExample
{
```

NOTES

```
public static void main(String[] args) throws IOException
{
//creating file output stream
FileOutputStream fos=new FileOutputStream
("employee.txt");
/*creating data output stream that filters file output
stream*/
DataOutputStream dos=new DataOutputStream(fos);
int empID[]={1,2,3,4};
String empName[]={"Maria","James","Mark","Renne"};
    float empSalary[]={50000,60000,70000,30000};
    //writing primitive data to the file
System.out.println("Writing to the file...");
    for (int i=0; i<empID.length; i++)
{
dos.writeInt(empID[i]);
dos.writeUTF(empName[i]);
dos.writeFloat(empSalary[i]);
}
dos.close();
System.out.println("Write successful");
System.out.println();

FileInputStream fis=new FileInputStream ("employee.txt");
//creating file input stream
/*creating data input stream that wraps file input stream*/
DataInputStream dis=new DataInputStream(fis);

//reading data from the file
System.out.println("Reading the file...");
try
{
while(true)
{
int empID1=dis.readInt();
String empName1=dis.readUTF();
float empSalary1=dis.readFloat();
System.out.println("Employee id: "+empID1);
```

```
System.out.println("Employee name: "+empName1);
System.out.println("Employee salary: "+empSalary1);
}
}
catch(Exception e){}
System.out.println();
System.out.println("Read successful");
}
}
```

NOTES

Output of the program:

```
Writing to the file...
Write successful
Reading the file...
Employee id: 1
Employee name: Maria
Employee salary: 50000.0
Employee id: 2
Employee name: James
Employee salary: 60000.0
Employee id: 3
Employee name: Mark
Employee salary: 70000.0
Employee id: 4
Employee name: Alice
Employee salary: 30000.0
Read successful
```

In this program, a file `employee.txt` is created. Data is written to this file using `DataOutputStream` class. After that, an input stream is created using `DataInputStream` class for reading the content of this file.

14.5 THE `StringTokenizer` CLASS

The `StringTokenizer` class, a part of `java.util` package, defines methods to parse an input string. Parsing a string requires breaking the string into small parts known as **tokens**. The breaking up of the string into tokens is determined by the **delimiters**. Delimiters are the characters that separate tokens within the string. There are two types of delimiters: *Single-character delimiters* and *Multi-*

NOTES

character delimiters. The single-character delimiters comprises of only one character, such as comma, white space, semicolon, colon, etc. For example, the tokens of the string `"What is your name?"` will be `"What"`, `"is"`, `"your"`, `"name?"` determined by the delimiter "white space". On the other hand, a multi-character delimiter consists of a sequence of characters and each character in the sequence is considered as a delimiter. For example, ",`*:_`" is a multi-character delimiter which splits the string into tokens based on the occurrence of comma, asterisk, colon and underscore.

The `StringTokenizer` class defines the following types of constructors.

```
StringTokenizer(String string)      //first
StringTokenizer(String string, String delimiters)
//second
StringTokenizer(String string, String delimiters, boolean
returndelimiters)         //third
```

The first constructor creates a string tokenizer for the `string` that is to be tokenized (parsed) and splits it into tokens based on the default delimiters. The default delimiters defined by Java are blank space, newline, carriage return and tab.

The second constructor tokenizes the `string` depending upon the `delimiters` specified as the second argument in the constructor.

The third constructor breaks the `string` into tokens on the basis of `delimiters`; in addition, it returns the delimiter characters that separate the tokens if the `returndelimiters` is set to `true`.

Some of the methods defined by the `StringTokenizer` class are listed in Table 14.3.

Table 14.3 *`StringTokenizer` Class Methods*

Method	Description
`String nextToken()`	returns the next available token in the string
`String nextToken(String delimiters)`	returns the next token in the string and changes the previously set delimiters to the set of characters in the string `delimiters`. After this method is invoked, `delimiters` is considered as the new delimiter set
`boolean hasMoreTokens()`	checks whether or not tokens are available in the input string. It returns `true` if `tokens are present. Otherwise, returns false`
`int countTokens()`	returns the number of tokens left in the string using the current set of delimiters
`Object nextElement()`	returns the same value as the `nextToken()` method except that it returns next token as object
`boolean hasMoreElements()`	returns the same value as the `hasMoreTokens()` method

NOTES

Program 6: A program to demonstrate the use of `StringTokenizer` class.

```
import java.io.*;
import java.util.*;
class StringTokenizerExample
{
    public static void main(String  args[])
    {
String  input  =”The  Blue  Hor,izon”;  /*string  to  be
tokenized*/
       String delimiters=”, n”;//specifying delimiters
StringTokenizer s=new StringTokenizer(input,delimiters
,true);
System.out.println(“Total number of tokens in the string:
“+s.countTokens());
       System.out.println();
       int n=0;
       System.out.println(“The tokenized string is: “);
       while (s.hasMoreElements())       //parsing  the
string
       {
System.out.println((++n)+”: “+s.nextElement());
}
}
}
```

Output of the program:

```
Total number of tokens in the string is: 8
  The tokenized string is:
  1: The
  2:
  3: Blue
  4:
  5: Hor
  6: ,
  7: izo
  8: n
```

14.6 THE `StreamTokenizer` CLASS

NOTES

The `StreamTokenizer` class is used to parse an input stream into 'tokens' determined by the delimiters, reading them one at a time. Unlike `StringTokenizer` class, the `StreamTokenizer` class distinguishes among numbers, identifiers, quoted strings and different comment styles. The tokenization process involves reading each byte from the input stream and each byte is considered as a character having five distinct attributes, namely, *alphabets*, *numbers*, *string quote*, *white space* and *comment character*. Each character is identified to be having either of these attributes.

The `StreamTokenizer` class defines the following type of constructor:

```
StreamTokenizer(Reader input)
```

This statement constructs a stream tokenizer and tokenizes the specified input stream `input`.

Some of the fields defined by the `StreamTokenizer` class are listed in Table 14.4.

Table 14.4 *`StreamTokenizer` Class Fields*

Field	Description
`static int TT_WORD`	indicates that the word token has been read
`static int TT_NUMBER`	indicates that the number token has been read
`static int TT_EOL`	indicates that the end of the line has been read
`static int TT_EOF`	indicates that the end of the file has been read
`int ttype`	specifies the type of the token read
`double nval`	the value of number token
`String sval`	specifies the value of a word token

Some of the methods defined by the `StreamTokenizer` class are listed in Table 14.5.

Table 14.5 *`StreamTokenizer` Class Methods*

Method	Description
`int lineno()`	returns the current line number
`void eolIsSignificant (boolean flag)`	determines whether the ends of the line are treated as tokens
`int nextToken()`	returns the next token of the input stream
`void whitespaceChars (int low, int high)`	specifies that all the characters within the range between `low` and `high` are white space characters
`void wordChars(int low, int high)`	specifies that all the characters within the range between low and high are valid word characters.
`void slashSlashComments (boolean flag)`	sets whether the stream tokenizer recognizes C++ comments
`void slashStarComments (boolean flag)`	sets whether the stream tokenizer recognizes C comments
`void resetSyntax()`	resets the default set of delimiters

NOTES

To understand the concept of `StreamTokenizer` class, we use the file `student.txt` having its contents as follows:

```
Name: Amit Sharma
Roll No.: 1
Overall Percentage: 89%

//Name: Shivani Dutta
Roll No.: 2
Overall Percentage: 92%
```

Program 7: A program that reads from the file `student.txt` to demonstrate the use of `StreamTokenizer` class.

```
import java.io.*;
class StreamTokenizerExample
{
   public static void main(String args[]) throws
IOException
   {

      int numCount=0, wordCount=0, lineCount=0;
//Creating the tokenizer to read from the file
      FileReader f=new FileReader("student.txt");
      StreamTokenizer s=new StreamTokenizer(f);
s.eolIsSignificant(true);/*end of line is considered
 as tokens*/
      //these calls will make the comments recognizable
      s.slashSlashComments(true);
      s.slashStarComments(true);

/*all the characters between this range(A to F) will be
considered as white spaces*/
      s.whitespaceChars(65,70);

      //tokenizing file
      int token=s.nextToken();
      while(token!=StreamTokenizer.TT_EOF)
      {
         switch(token)
         {
```

NOTES

```
            case StreamTokenizer.TT_NUMBER:
            double num=s.nval;
            numCount++;
System.out.println("Number token is: "+num);
      break;

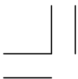
            case StreamTokenizer.TT_WORD:
            String word=s.sval;
            wordCount++;
      System.out.println("Word token is: "+word);
      break;

      case StreamTokenizer.TT_EOL:
            lineCount++;
System.out.println("End of line number " +lineCount);
      break;

      default:
      char ch=(char)s.ttype;
      break;
}
   token=s.nextToken();
}
System.out.println("Total number of number tokens is:
"+numCount);
System.out.println("Total number of word tokens is:
"+wordCount);
System.out.println("Total number of lines is: "+
lineCount);
      f.close();
   }
}
```

Output of the program:

```
Word token is: Name
Word token is: mit
Word token is: Sharma
End of line number 1
Word token is: Roll
```




NOTES

```
Word token is: No.
Number token is: 1.0
End of line number 2
Word token is: Overall
Word token is: Percentage
Number token is: 89.0
End of line number 3
End of line number 4
End of line number 5
Word token is: Roll
Word token is: No.
Number token is: 2.0
End of line number 6
Word token is: Overall
Word token is: Percentage
Number token is: 92.0
End of line number 7
Total number of number tokens is: 4
Total number of word tokens is: 11
Total number of lines is: 7
```

14.7 THE `TRANSIENT` AND `VOLATILE` MODIFIERS

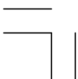
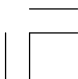
The transient and volatile modifiers are two special modifiers provided by Java which are used to handle some specialized situations.

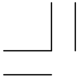
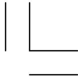
`transient`

Object serialization is the process of reading and writing objects. By default, all objects are serializable, i.e., they can be read from and written to the secondary memory so that the value which they hold persists. To make an object non-serializable, the `transient` modifier is used. If an instance variable is declared as transient then the values of that variable will not persist while writing its object to the secondary memory.

To understand the concept of transient keyword, consider the following code segment:

```
class TransientExample
{
    transient double first;         // will not persist
    double second;                  //will persist
}
```

Here, when the object of class `TransientExample` is written to the secondary memory, the values of `first` will not be saved while the value of `second` will be.

NOTES

`volatile`

The `volatile` modifier is used to tell the compiler that the variable declared as `volatile` can be changed at any time by the other parts of the program. This modifier is mainly used in multithreading in which a program (process) is divided into two or more subprograms (subprocesses), each of which runs by a separate thread and performs different tasks concurrently. In a multithreaded program, various threads share the same instance variable and keep their own copy of the variable in their local cache memory while the master copy remains in the main memory. Whenever a thread changes the value of the variable, it updates the value of the variable only in its local cache memory and not in the main memory. This leads to inconsistency because the thread using the same variable does not know about the change of the value by another thread. So to avoid this problem, the `volatile` modifier is used. When a variable is declared as `volatile`, it is not stored in the cache memory and its value is updated in the main memory so that the other threads can easily access the updated value.

14.8 USING `INSTANCEOF` OPERATOR

The `instanceof` operator is an operator which is used to check whether the object belongs to a particular class or not.

The general form of `instanceof` operator is as follows:

```
object instanceof type
```

where,

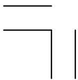
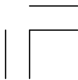
`object` is an instance of the class

`type` is the class type

The `instanceof` operator returns `true` if the `object` represents an instance of specified `type` or can be cast into a specified `type`, otherwise it returns `false`.

Program 8: A program to demonstrate the use of `instanceof` operator.

```
class FirstClass
{
    void show()
    {
        System.out.println("Method of FirstClass");
    }
```

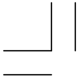
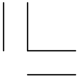
NOTES

```
}
class SecondClass extends FirstClass
{
   void show()
   {
      System.out.println("Method of SecondClass");
   }
   void showstatus()
   {
      System.out.println("Another method of SecondClass");
   }
}
public class InstanceofExample
{
   public static void main(String args[])
   {

      FirstClass fc = new FirstClass();

      SecondClass sc = new SecondClass();
      if(fc instanceof FirstClass)
      {
         System.out.println("fc is an instance of
         FirstClass");
         fc.show();
      }

      if(sc instanceof SecondClass)
      {
         System.out.println("sc is an instance of
         SecondClass");
         sc.show();
         sc.showstatus();
      }
      else
      {
         System.out.println("sc is not an instance of
      SecondClass");
      }
   }
}
```

Output of the program:

```
fc is an instance of FirstClass
Method of FirstClass
sc is an instance of SecondClass
Method of SecondClass
Another method of SecondClass
```

14.9 NATIVE METHODS

A native method is a special method which contains a native code written in any programming language other than Java. Generally, the native code is written in C language. A method is declared native by preceding the method with the keyword `native` with no body defined.

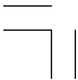
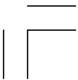
Program 9: Let us consider an example to understand the concept of native methods. The following steps show how the native methods are created and used in Java.

1. A native method `show()` is declared in a normal Java class called `NativeExample`.

```
public class NativeExample
{
 public native String show(String message);
 public static void main(String args[])
 {
    String message = null;
    NativeExample ne = new NativeExample();
    message = ne.show("I am called through a native
       method");
    System.out.println(message);
 }
}
```

2. The native implementation of `show()` is contained in **dynamic link library** which is loaded using the `static` method. The `static` method is executed only once when the execution of the program begins. The code which is used to load the dynamic link library is

```
static
{
 System.loadLibrary("NativeExample");
}
```

NOTES

```
The dynamic link library is loaded by using the
loadLibrary() method of System class.

The general form of loadLibrary() is
static void loadLibrary(String filename)

where,
filename specifies the name of the file that holds
the library

The complete Java file called NativeExample is  as
follows:
public class NativeExample
{
 public native String show(String message);
 static
 {
    System.loadLibrary("NativeExample");
 }
 public static void main(String args[])
 {
    String message = null;
    NativeExample ne = new NativeExample();
    message = ne.show("I am called through a native
       method");
    System.out.println(message);
 }
}
```

3. The Java file `NativeExample` is compiled to produce its class file `NativeExample.class` by using the command given below.

```
javac NativeExample.java
```

4. The `.h` file of `NativeExample` is generated by using the command given below.

```
javah -jni NativeExample
```

This command generates a header file `NativeExample.h` which has the following code.

```
/* DO NOT EDIT THIS FILE
- it is machine generated */
#include <jni.h>
/* Header for class NativeExample */
#ifndef _Included_NativeExample
#define _Included_NativeExample
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     NativeExample
 * Method:    show
 * Signature: (Ljava/lang/String;)V
 */
JNIEXPORT void JNICALL Java_NativeExample_show (JNIEnv
*, jobject, jstring);
#ifdef __cplusplus
}
#endif
#endif
```

NOTES

In this code, the following statement defines the prototype for the native method `show()` which is to be created.

```
JNIEXPORT void JNICALL Java_NativeExample_show (JNIEnv
*, jobject, jstring);
```

where,

`JNIEXPORT` and `JNICALL` represent the macros that expand to match platform specific directives.

`JNIEnv`, `jobject` and `jstring` represent the JNI data type definitions.

`Java_NativeExample_show` is the name of the method which is used during the implementation of the native method. The general form of this method is as follows:

```
Java_className_methodName
```

NOTES

where,

`Java` is added as a prefix.

`className` is the name of the Java class in which the native method is declared. It is added as a prefix to identify the associated native method.

`methodName` is the name of the native method.

Since the name of native method is appended with the class name, different Java classes can declare a native method with the same name.

5. A file called `NativeExample.c` is created which implements the method `show()`. The code for `NativeExample.c` is as follows:

```
#include<jni.h>
#include "NativeExample.h"
#include<stdio.h>
JNIEXPORT void JNICALL Java_NativeExample_show (JNIEnv
*env, jobject obj, jstring msg)
{
 return msg;
}
```

6. Compile the file `NativeExample.c` and create a DLL file called `NativeExample.dll`.
7. The java program is run using the following command

```
java NativeExample
```

Output of the program:

```
I am called through a native method
```

Check Your Progress

1. To write the data to the console, which stream is used?
2. What is a file?
3. Why `FileInputStream` class is used in the program.
4. What does data streams provide?
5. Why the `StringTokenizer` class is used?
6. What does tokenization process involves?
7. Explain about the transient and volatile modifiers provided by Java.
8. What are native methods?

14.10 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. To write the data to the console, the `System.out` stream is used which belongs to the `PrintStream` class. The `PrintStream` class defines two methods to write output to console, one is `print()` and the other is `println()`. Both methods perform the same task except that the `println()` method writes the output in a new line while `print()` method writes the output in the same line.
2. A **file** is a collection of related data stored on some storage device. The data stored in a data file is permanent and can easily be accessed as and when required.
3. The `FileInputStream` class creates an input stream between the file and the program and reads bytes from the file and sends it to the program.
4. Data streams support reading/writing of primitive data types (`int`, `float`, `short`, `long`, `double`, `boolean`, `char` and `byte`) as well as `String` objects. These are required when the user wants to work on data other than bytes or characters. Data streams are filtered streams, that is, they filter existing stream so that the primitive data types can be directly read from or written to the stream. Java provides `DataInputStream` class and `DataOutputStream` class to read and write primitive data types, respectively.
5. The `StringTokenizer` class, a part of `java.util` package, defines methods to parse an input string. Parsing a string requires breaking the string into small parts known as **tokens**. The breaking up of the string into tokens is determined by the **delimiters**. Delimiters are the characters that separate tokens within the string. There are two types of delimiters: *Single-character delimiters* and *Multi-character delimiters*.
6. The tokenization process involves reading each byte from the input stream and each byte is considered as a character having five distinct attributes, namely, *alphabets*, *numbers*, *string quote*, *white space* and *comment character*. Each character is identified to be having either of these attributes.
7. The transient and volatile modifiers are two special modifiers provided by Java which are used to handle some specialized situations.
8. A native method is a special method which contains a native code written in any programming language other than Java. Generally, the native code is written in C language. A method is declared native by preceding the method with the keyword `native` with no body defined.

14.11 SUMMARY

NOTES

- Java 1.0 supports console input using byte streams. Since the use of byte streams for reading/writing console input requires using deprecated methods, this approach is not recommended.
- In order to read data from the console using character stream, `System.in` is used. As `System.in` refers to an object of type `InputStream`, it can be used as a character-based input stream.
- To write the data to the console, the `System.out` stream is used which belongs to the `PrintStream` class. The `PrintStream` class defines two methods to write output to console, one is `print()` and the other is `println()`. Both methods perform the same task except that the `println()` method writes the output in a new line while `print()` method writes the output in the same line.
- A **file** is a collection of related data stored on some storage device. The data stored in a data file is permanent and can easily be accessed as and when required.
- For reading and writing data to the files, file streams are used. A file stream refers to the flow of data between the program and the files (see Figure 14.1). Depending on the type of data handled by the file stream, byte stream classes or character stream classes can be used.
- Before reading or writing a file, it must be opened first. For this, it is required to create a file stream and then link this stream to the specific file. Java provides `FileInputStream` class to read bytes from a file.
- Data streams support reading/writing of primitive data types (`int`, `float`, `short`, `long`, `double`, `boolean`, `char` and `byte`) as well as `String` objects. These are required when the user wants to work on data other than bytes or characters. Data streams are filtered streams, that is, they filter existing stream so that the primitive data types can be directly read from or written to the stream.
- The `DataInputSream` class performs reading of Java primitive data types from the input stream.
- The `StringTokenizer` class, a part of `java.util` package, defines methods to parse an input string. Parsing a string requires breaking the string into small parts known as **tokens**.
- The `StreamTokenizer` class is used to parse an input stream into 'tokens' determined by the delimiters, reading them one at a time.
- The transient and volatile modifiers are two special modifiers provided by Java which are used to handle some specialized situations.

- The `volatile` modifier is used to tell the compiler that the variable declared as `volatile` can be changed at any time by the other parts of the program.
- The `instanceof` operator is an operator which is used to check whether the object belongs to a particular class or not.
- A native method is a special method which contains a native code written in any programming language other than Java. Generally, the native code is written in C language. A method is declared native by preceding the method with the keyword `native` with no body defined.

NOTES

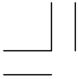
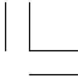
14.12 KEY WORDS

- **File:** A file is a collection of related data stored on some storage device. The data stored in a data file is permanent and can easily be accessed as and when required.
- **Data streams:** Data streams support reading/writing of primitive data types (`int`, `float`, `short`, `long`, `double`, `Boolean`, `char` and `byte`) as well as `String` objects. These are required when the user wants to work on data other than bytes or characters.
- **Tokens:** Parsing a string requires breaking the string into small parts known as tokens.
- **Delimiters:** The breaking up of the string into tokens is determined by the delimiters, these are the characters that separate tokens within the string.
- **Object serialization:** Object serialization is the process of reading and writing objects. By default, all objects are serializable, i.e., they can be read from and written to the secondary memory so that the value which they hold persists.

14.13 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What are I/O classes?
2. Why I/O classes uses stream?
3. Define data streams in Java.
4. How reading and writing files is done in Java?
5. Explain the process of reading writing bytes in Java.
6. Why is native method used?

NOTES

Long Answer Questions

1. Briefly explain the concept of I/O classes using stream in Java giving appropriate examples.
2. Discuss the concept of reading from and writing to console in Java language.
3. Discuss all the reading and writing files methods giving appropriate examples in Java.
4. Write a Java program for reading writing bytes in Java.
5. Write a program in Java to illustrate the use of `System.out` and `System.in`.
6. Write a program in Java to demonstrate the use of `DataInputStream` class and `DataOutputStream` class for handling primitive data types.

14.14 FURTHER READINGS

Krishnamoorty, R. and Prabhu R. Krishnamoorty. 2009. *Internet and Java Programming*. New Delhi: New Age International (P) Ltd.

Balagurusamy, E. 2007. *Programming with Java*, Third Edition. New Delhi: Tata McGraw-Hill.

Das, Rashmi Kant. 2009. *Core Java for Beginners*, Revised Edition. New Delhi: Vikas Publishing House Pvt. Ltd.

Keogh, Jim. 2002. *The Complete Reference J2SE*, Fifth Edition. New York: Tata McGraw-Hill.

Naughton, Patrick and Herbert Schidt. 1999. *Java 2: The Complete Reference*, Third Edition. New Delhi: Tata McGraw-Hill.

